Springer Series in Reliability Engineering

P.K. Kapur · H. Pham · A. Gupta · P.C. Jha

Software Reliability Assessment with OR Applications

Springer

Springer Series in Reliability Engineering

For further volumes:
http://www.springer.com/series/6917

P. K. Kapur · H. Pham · A. Gupta · P. C. Jha

Software Reliability Assessment with OR Applications

Springer

Prof. P. K. Kapur
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: pkkapur1@gmail.com

Dr. A. Gupta
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: guptaanshu.or@gmail.com

Prof. H. Pham
Department of Industrial and Systems
Engineering
Rutgers University
Frelinghuysen Road 96
Piscataway, NJ 08854-8018
USA
e-mail: hopham@rci.rutgers.edu

Dr. P. C. Jha
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: jhapc@yahoo.com

ISSN 1614-7839

ISBN 978-0-85729-203-2 e-ISBN 978-0-85729-204-9

DOI 10.1007/978-0-85729-204-9

Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

© Springer-Verlag London Limited 2011

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 licenses 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.

Cover design: eStudio Calamar, Berlin/Figueres

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Advances in software technologies have promoted the growth of computer-related applications to a great extent. The proliferation of Internet has gone far beyond even the most optimistic forecasts. Computers and computer-based systems pervade every aspect of our daily lives. This has benefited society and increased our productivity, but it has also made our lives critically dependent on their correct functioning. Successful operation of any computer system depends largely on its software components. In the past three decades abilities to design, test and maintain software have grown fairly, but the size and design complexities of the software have also increased manyfolds, and the trend will certainly continue in future. In addition to this, the critical system operations, in which very high operational precision is required are also becoming more and more dependent on the software. There are numerous instances where failure of computer-controlled systems has led to colossal loss of human lives and money. This is a big challenge to the software developers and engineers. Producing and maintaining the high quality of software products and processes are at the core of software engineering, and only a comprehensive quality improvement and assessment program that have successful outcome can assure it. A lot of research material and book titles are available with focus on tools and methods for monitoring and assuring high quality software. At this stage there is a great need for looking at ways to quantify and predict the reliability of software systems in various complex embedded operating systems. Apart from this, cost and budget limitations, schedule, and due dates are the constraints that encroach on the degree to which software development and maintenance professional can achieve maximum quality. Our title *Software Reliability Assessment with OR Applications* provides in-depth knowledge of quantitative techniques for software quality assessment.

The technology of modern embedded software systems is changing at a very fast rate; such changes are not ever seen in any other areas. On account of these changes, the techniques and models available to measure the system reliability have also increased at the same rate. In contrast to the few available books in this area our book addresses most of the existing research, recent trends, and many more of these techniques and models. Several areas of software reliability

assessment and applications, which have gained interest mainly in the last five years and grown at a very fast pace, have been discussed comprehensively in the book for the first time. Topics such as

- Change point models in software reliability measurement
- Application of neural networks to software reliability assessment
- Optimization problems of optimum component selection in fault tolerant systems
- Unification methodologies in software reliability assessment
- Software reliability growth modeling using stochastic differential equationshave been included for first time, while topics such as
- Literature of reliability analysis for fault tolerant systems
- Study of software release time decision
- Optimum resource allocation problem

have been addressed comprehensively.

The content of this book is useful and provides solution to the problems faced by several groups of people working in the different fields of software industry.

These groups in general are the people

1. Who want to acquire the knowledge of the state-of-the-art of software reliability measurement, prediction and control. These people include the managers of the software development organizations, engineering professional dealing with software, and persons involved in the marketing and use of software.
2. Who are working in different software development groups such as software design team, testing and debugging teams, and maintenance and evolution teams, or practitioners of quality assessment, risk analysis, management, and decision sciences.
3. Who are involved in the research related to software reliability engineering, reliability analysis, operations research, applied statistics and mathematics, and industrial engineering and related disciplines.

The book brings out widespread literature of past 40 years of software reliability assessment. It can serve as a first choice and a complete reference guide. The book brings out widespread literature of past 40 years of software reliability assessment. It can serve as a first choice and a complete reference guide

The introduction chapter provides an inclusive material and basic knowledge required to understand the entire content of the book. Various new concept maps and pictures have been designed to facilitate the understanding. The content of rest of the book is organized as follows.

Chapter 2 describes the earlier literature of the software reliability growth models (SRGM). It covers the software reliability modeling with exponential, S-shaped and flexible models. Consideration of testing efforts in reliability growth modeling is also presented. The last section of the chapter concentrates on reliability assessment models for software developed under distributed environment.

Earlier literature of reliability growth modeling assumed a perfect debugging environment. Testing the efficiency of testing and debugging teams makes an important aspect of the reliability growth modeling, and its consideration in the models can give absolutely different results as compared to perfect debugging models. The literature of software reliability modeling under imperfect debugging environment is discussed in Chapter 3.

Testing coverage and testing domain measures are the key factors related to the software reliability growth process. These measures help developers to evaluate the quality of the tested software and determine the additional testing required to achieve the desired reliability. On the other hand, it is a quantitative confidence criterion for the customer in taking the decision to buy the product. A detailed discussion on the testing coverage, domain, and reliability modeling with respect to these measures is done in Chapter 4.

The concept of change point is relatively recent in the software reliability modeling. Developing models using the change point concept provides very accurate results most of the times. A number of reasons are associated for modeling under change point concept such as changes in the testing environment, testing strategy, complexity and size of the functions under testing, defect density, skill, and motivation and constitution of the testing and debugging team. Modeling using the change point concept provides answers to the number of questions related to the changing scenarios during testing phase. Reliability modeling with change point is discussed at length in Chapter 5.

Chapter 6 is addressed to the unification schemes in software reliability growth modeling. Several existing SRGM consider one or the other aspect of software testing but none can describe a general testing scenario. As such, for any particular practical application of reliability analysis one needs to study several models and then decide the most appropriate one. The selected models are compared based on the results obtained and then a model is selected for further use. As an alternative, following a unification approach several SRGM can be obtained from a single approach giving an insightful investigation of these models without making many distinctive assumptions. It can make our task of model selection and application much simpler compared to the other methods. Establishment of unification methodology is one of the very recent topics of research in software reliability modeling and is discussed for the first time in this book.

Like unification schemes, software reliability modeling based on the Artificial Neural Networks has gained interest of software reliability researchers recently. Only limited work has been done in the field by a group of few researchers. In Chapter 7 we introduce and discuss the existing literature in this area.

The topic of software reliability modeling with stochastic differential equations although started in the early nineties but gained much popularity and seen more useful work only in the current years. A comprehensive study of this topic is presented in Chapter 8.

The reliability growth models discussed in the previous chapters are the continuous time models. There is another category of reliability growth models, which use the test cases as a unit of fault detection/removal period. These models are

called discrete time models. A large number of models have been developed in the first group while fewer are there in the second group due to the difficulties in terms of mathematical complexity. The utility of discrete reliability growth models cannot be underestimated. As the software failure data sets are discrete, these models many a time provide better fit than their continuous time counterparts. Chapter 9 addresses to the study of discrete software reliability growth modeling.

The software reliability models find important OR applications. Determination of software release time and allocation of testing resources at unit testing level are among the major applications. Chapters 10 and 11 present an inclusive study of these optimization applications of the reliability growth models.

Maintaining highest possible reliability is most important for the software systems used to automate the critical operations. Fault tolerance is designed in software to achieve the highest level of reliability in these systems as compared to what can be attained with testing. A complete knowledge of fault tolerant schemes, reliability growth modeling, and optimum system composition problem has been described in Chapter 12.

A number of useful references, appendices, and index terms are provided to help further readings.

We expect that our book will meet the expectations of the readers and provide the best of the state-of-the-art on the subject.

Acknowledgments

Prof. Kapur remembers with special fondness, the first in person meeting with Prof. Pham at the International Conference on *Present Practices and Future Trends in Quality and Reliability*, Indian Statistical Institute, Kolkata, in 2008. They shared thoughts together and their thoughts conceptualized the idea of this book. Later Prof. Kapur associated Dr. Jha and Dr. Gupta, who happen to be his former students with his ideas and the concept came into the form of a proposal. It took almost two years to complete the book and it is the time to have the opportunity to acknowledge the people who provided their support directly or indirectly in completing this venture.

This book contains a lot of research material of various researchers across the globe of over nearly four decades and more. The list of authors whose contributions have been incorporated in this book is very big. It was not possible to specifically list them all individually, the authors of the book like to greatly acknowledge their outstanding contributions, appreciate their work, and thank all of them.

Prof. Kapur and Prof. Pham are also grateful to their numerous Ph.D\M.Tech\M.Phil\fellow students for the research work, done jointly with them. Their contribution is immeasurable to the growth of this book.

The authors wish to express deep sense of gratitude to their parents, spouses, children, and other family members, who have provided them unconditional and unfaltering support. Their supportive attitude was always motivating.

We are thankful to almighty God for giving us the strength to complete our work.

Lastly, we apologize for any omissions.

Contents

Acronym

ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
CDF	Cumulative Density Function
CER	Community Error Recovery
CF	Common Faults
CFM	Common Failure Mode
CIF	Concurrent Independent Failures
CIFM	Concurrent Independent Failure Mode
CLNLR	Conditional Logic Nonlinear Regression
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRB	Consensus Recovery Block
DDE	Distributed Development Environment
DIM	Dynamic Integrated Model
DNA	Deoxyribonucleic Acid
DWCM	Dynamic Weighted Combinational Model
EDRB	Extended Distributed Recovery Block
ENHPP	Enhanced Non-Homogeneous Poisson Process
EW	Error Derivative of the Weights
FCP	Fault Correction Process
FDP	Fault Detection Process
FDR	Fault Detection Rate
FIR	Fault Isolation Rate
FRR	Fault Removal Rate
GDIM	Generalized Dynamic Integrated SRGM
GINHPP	Generalized Imperfect Non-Homogeneous Poisson Process
GO Model	Goel and Okumoto Model
GOS	Generalized Order Statistic
GPA	Goal Programming Approach
HPP	Homogeneous Poisson Process

IBM	International Business Machines
IEEE	Institute for Electrical and Electronic Engineers
IF	Independent Faults
ISO/IEC	International Organization for Standardization/International Electro-Technical Commission
IT	Information Technology
KG Model	Kapur and Garg Model
KLOC	Kilo Lines of Code
KT	Kuhn Tucker
LMS	Least Mean Squares
LN	Levenberg-Marquardt
METEF	Modified Exponential Testing Effort Function
MLE	Maximum Likelihood Estimate
MOV	Modified Optimal Values
MRTEF	Modified Rayleigh Testing Effort Function
MSE	Mean Square Error
MTBF	Mean Time between Failures
MTTF	Mean Time to Failure
MWTEF	Modified Weibull Testing Effort Function
NDP	Normalized Detectability Profile
NHPP	Non-Homogeneous Poisson Process
NLLS	Non-Linear Least Square
NLR	Nonlinear Regression
NMR	N-Modular Programming
NN	Neural Network
NVP	N Version Programming
OOV	Original Optimal Values
OS	Operating System
PDF	Probability Density Function
PE	Prediction Error
PGF	Probability Generating Function
R&D	Research and Development
RB	Recovery Blocks
RMSPE	Root Mean Square Prediction Error
RPE	Relative Predictive Error
SDE	Stochastic Differential Equations
SDLC	Software Development Life Cycle Models
SPSS	Statistical Package for Social Sciences
SQP	Sequential Quadratic Programming
SRE	Software Reliability Engineering
SRGM	Software Reliability Growth Model
SRTD	Software Release Time Decision
TEF	Testing Effort Function

TFN	Triangular Fuzzy Number
V&V	Verify and Validate
WRC	Water Reservoir Control
YDSM	Yamada Delayed S-Shaped Model

Chapter 1
Introduction

A popular theory and explanation of the contemporary changes occurring around us is that we are in the midst of a third major revolution in human civilization, i.e., a *Third Wave*. First there was the Agricultural Revolution, then the *Industrial Revolution*, and now we are in the *Information Revolution*. Yet we are, in fact, in the middle of a revolutionary jump. Information and communication technology and a worldwide system of information exchange have been growing for over a 100 years. Information technology (IT) is playing a crucial role in contemporary society. It has transformed the whole world into a global village with a global economy. IT has now become the most important technology in the human world and it is an excellent example of the *law of unintended consequences* as it paves the way for creation of the new technologies (e.g., genetic engineering), extension of the existing technologies (e.g., telecommunications), and the demise of the older technologies (e.g., the printing industry). Today almost every business, industry, services, government agencies, and even our day-to-day activities are directly or indirectly affected by computing systems. The Computer revolution has benefited society and increased the global productivity, but a major threat of this revolution is that the world has become critically dependent on the computing systems for the proper functioning and timing of all its activities. For example, air traffic control, nuclear reactors, patient monitoring system in hospitals, automotive mechanical and safety control, online railways and air ticketing, industrial process control, global networking of various business, and services which include information storing (databases), information sharing and internet marketing, etc. are some diverse applications of IT. If the computer system shows a failure in such systems the impact of failures may range from inconvenience in social life to economic damage to loss of life in the most critical case. A total breakdown of the system functioning is observed in most of the cases until the fault is repaired, and even after restoring the system to a normal state, sometime it takes up huge time, efforts, and resources to make up the losses.

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_1,
© Springer-Verlag London Limited 2011

In the broadest sense, IT refers to both the hardware and software that are used to store, retrieve, and manipulate information. In the past two decades the hardware has attained high productivity and quality due to advances in technology and progress of design and test mechanisms. Large-scale improvement in hardware performance, profound changes in computing architectures, vast increase in memory and storage capacity, a wide variety of exotic input and output options, has further increased the demand of software in automation of complex systems, its use as a problem-solving tool for many complex problems of exponential size, and to control critical applications. With this, size and design complexities of the software has also increased many folds and the trend will certainly continue in future. For instance the NASA Space Shuttle flies with approximately 0.5 million lines of software code on board and 3.5 million lines of code in ground control and processing [1–3].

With the escalation in size, complexity, demand, and depends on the computer systems the risk of crises due to software failures has also increased. There are numerous reported and unreported instances when software failures have caused severe losses [3, 4]. Few examples are the crash of Boeing 727 of Mexicana airlines because the software system did not correctly negotiate the mountain position (1986), overdose given to the cancer patients by the massive Therac-25 radiation machine in Marietta due to flaws in the computer program controlling the device (1985 and 1986), Explosion of the European Space Agency's Ariane 5 rocket, in less than 40 s after lift-off on 4 June 1996 due to software design errors and insufficient testing, blackouts in the North-East US during the month August, 2003 due to an error in the AEPR (Alarm and Event Processing Routine) software, etc.

The abilities to design, test, and maintain software has grown fairly, lot of further improvements are desired in the field. The software development process has become really a challenging task for the developers. Accordingly, the main concern about productivity and quality of computer systems has been changing from the hardware to the software systems. Now the question arises, what makes *productive and quality software*? The answer is, the software that enables a seamless technology experience for people wherever they are—in the home, in the office or on the go. Arguably the most important software development problem is building software to customer demands so that it will be *more reliable, built faster*, and *built cheaper* (in general order of importance) [5]. Success in meeting these demands affects the market share and profitability of a product for the developer. These demands conflict, causing risk and overwhelming pressure, and hence strong need for a practice that can help them to have a tight control over the software development process and develop software to the need of the software market.

Software reliability engineering (SRE) discipline came forward to create and utilize sound engineering principles in order to economically obtain software systems that are not only reliable but also work proficiently on real machines, in the early 1970s. This made software reliability study recognized as an engineering discipline. The next concern of software engineering was scheduling and systematizing the software development process to monitor the progress of the

various stages of software development using its tools, methods, and process to engineer quality software and maintaining a tight control throughout the development process. Here the most important thing that must be clearly defined is what quality refers to in context to the developers and the end users. More often it is defined in terms of internal quality and external quality with a focus on transforming the user's requirements (external quality characteristics) into the quality characteristics of the software system developers (internal quality characteristics). SRE broadly focuses on quantitatively characterizing the following standardized six quality characteristics defined by ISO/IEC: functionality, usability, reliability, efficiency, maintainability, and portability. *Software reliability* is accepted as the key characteristic of software quality since it quantifies software failures—the most unwanted events, and hence is of major concern to the software developers as well as user. Further it is the multidimensional property including other customer satisfaction factors such as functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. For this reason it is considered to be a "*must be quality*" of the software.

One of the major roles of SRE lies in assuring and measuring the reliability of the software. The tools of SRE known as software reliability growth models (SRGM) are used successfully to develop test cases, schedule status, to count the number of faults remaining in the software, and estimate and predict the reliability of the software during testing and operational environment. Foundation of research in reliability modeling for software seems to be as old as 40 years however, in the past 30 years the field has experienced extensive growth. Many reliability engineering scientists and research scholars have done excellent study of the various aspects of the software quality measurement during the software development and maintenance phases. Many SRE books are available that focus on software reliability modeling. However, similar to the IT advances, the software reliability modeling has also advanced, incorporating the many recent and challenging issues in reliability measurement. In this book we present the state-of-the-art modeling in software reliability over the past 40 years in one place. Various optimization applications of reliability modeling solved using the tools of operational research are also discussed in the later chapters of the book. In the next sections of this chapter we elaborate some important concepts of SRE, software development and testing, SRGM classification, and a brief review of the literature.

1.1 Software Reliability Engineering

Software engineering is relatively a young disciple and was first proposed in 1968 at a conference held to discuss the problem known at that time as software crisis. Software crisis was the result of introduction of the powerful, third-generation computer hardware. Many software projects run over budget and schedule, were unreliable, difficult to maintain, and performed poorly. The software crisis was originally defined in terms of productivity, but evolved to emphasize quality. New

techniques and methods were needed to control the complexities of the large software projects and the techniques developed and adopted lead to the foundation of SRE. The SRE technologies were mainly inherent (such as specification, design, coding, testing, and maintenance techniques) that aid in software development directly and management technologies (such as quality and performance assessment and project management) that support the development process indirectly. Our focus in this book lies on the management technologies.

A number of definitions of SRE are made by several people and it is difficult to say which definition describes it most appropriately. Here we would like to mention that the word engineering is an action word, which aims to find out ways to approach a problem. The problems as well as approaches to resolve them have changed drastically in the past decade and the changes are continued, the definition of SRE is also changing and evolving. The IEEE [6] society has defined SRE as widely applicable, standard, and proven practice that apply systematic, disciplined, quantifiable approach to the software development, test, operation, maintenance, and evolution with emphasis on reliability and the study in these approaches. Further ISO/IEC defined software reliability as "an attribute that a software system will perform its function without failure for a given period of time under specified operational environment".

There are several simultaneous benefits of applying SRE principles on any software development project; broadly they can be listed as—it insures that product reliability conforms to the user requirements, lowers the development cost and time with least maintenance and operation costs, improved customer satisfaction, increased productivity, reduced risk of product failure [5], etc. Conceptually SRE is a layered technology (Fig. 1.1). It rests on the organizational commitment to quality with a continuous process improvement culture and has its foundation in the process layer. Process defines the framework for management control of the software projects, establishes the context in which technical methods are applied, work products are produced, quality is insured, and change is properly managed. SRE methods provide the technical "how to's" for building the software whereas the tools provide automated or semi-automated support for the processes and methods [6].

SRE management techniques work by applying two fundamental ideas:

- Deliver the desired functionality more efficiently by quantitatively characterizing the expected use, use this information to optimize the resource usage focusing on the most used and/or critical functions, and make testing environment representative of operational environment.

Fig. 1.1 Software reliability engineering layers

- Balances customer needs for reliability, time, and cost-effectiveness. It works by setting quantitative reliability, schedule and cost objectives, and engineers' strategies to meet these objectives.

The activities of SRE include:

- Attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability.
- Software reliability measurement—estimation and prediction.
- The application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

There exist sound process models of SRE known as software development life cycle (SDLC) models, which describe the various stages of software development in a sequential and planned manner. Most of the models, model the SDLC in the following stages: requirement analysis and definition, system design, program design, coding, testing, system delivery, and maintenance. The tools and techniques of SRE provide means to the software engineer to monitor, control, and improve the software quality throughout the SDLC.

1.2 Software Development Life Cycle

Software development realized using the tools and techniques of SRE enables developers to deliver enough reliability avoiding both excessive costs and development time. Software development involves a set of ordered tasks; each task can be called as a generic process and the process of software development is known as SDLC. The IEEE computer dictionary has defined SDLC as "the period of time in which the software is conceived, developed and used". The software life cycle process model describes software products life from the conceptualization stage to the final implementation and maintenance stage. Many life cycle process models are described in the software engineering literature. The generic process framework applicable to the vast majority of software projects includes the following stages:

- Analysis and specification
- Software development
- Verification and validation
- Maintenance and evolution

Each framework activity is populated by a set of software engineering actions such as software project tracking and control, risk management, quality assurance and measurement, technical reviews, reusability measurement, etc. Following the generic framework activities every software development and engineering organization describes a unique set of activities it adopts with the complemented set of engineering actions in terms of a task set that identifies the work to be accomplished.

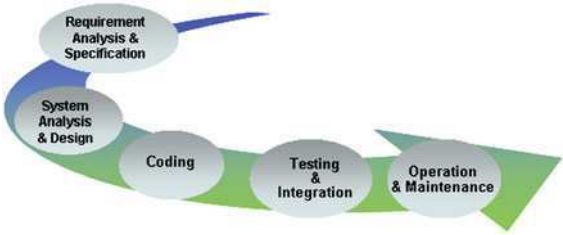


Fig. 1.2 Waterfall model

Almost all known process models bear at least some similarity to the preliminary process model known as waterfall models. The waterfall model was proposed by Royce in 1970. The framework activities of the model are shown in Fig. 1.2 and can be illustrated as follows.

Activity 1: Requirement Analysis and Specification

This phase forms the foundation stage for building successful software. Defining the project scope, software requirements, and providing specifications for the subsequent phases and activities. Project scope definition includes the study of the users' need for the system and their problems. This is accomplished with frequent interaction with the users. Once the scope is defined the requirement collection activity starts. Requirement collection is actually the study of product capabilities and constraints. It includes collection of product functionality, usability, intended use, future expectations, user environment, and operating constraints. Requirement analysis concludes with a feasibility study of user requirements, cost benefit estimation, and documentation of collected information and feasibility report. The document holds the different specific recommendations for the candidate system such as project proposal, environmental specifications and budget, schedule, and method plans. The immediate following activity is system specification. The basic aim of this activity is to transform the user requirement-oriented document to the developer-oriented document (design specifications). This is the first document that goes into the hands of the software engineers and forms the foundation document of the project; hence, it must precisely define essential system functions, performances, design constraints, attributes, and external interfaces. In this phase, the software's overall structure and its nuances are defined. All activities of this phase must be accomplished very crucially. A well-developed specification can reduce the occurrence of faults in the software and minimizes rework.

Activity 2: System Analysis and Design

System design activity is concerned with architectural and detailed project design. A detailed analysis of the specification document is carried to know the

performance, security and quality requirements, system assumptions, and constraints. This study enables partitioning of full system into smaller subsystems and definition of internal and external interface relationships. The needed hardware and software support are also identified. In terms of the client/server technology, the number of tiers needed for the package architecture, the database design, the data structure designs, etc. are all defined in this phase. The architectural design is completed with an architectural document design. This document is followed by a detailed system design activity. Here the program structure, algorithmic details, programming language and tools, test plans are specified. The final outcome of this phase is a detailed design document. The design engineers must take care that the designed system architecture, program structure, and algorithm design conforms to the specification document. Any glitch in the design phase could be very expensive to solve in the later stage of the software development.

Activity 3: Coding

The program structures and algorithms specified in the design document are coded in some programming language—a hardware readable form. This phase consists in identifying existing reusable modules, coding of new modules, modifications in existing modules, code editing, code inspection, and a final test plan preparation. If the program design is performed in a detailed manner, code implementation can be accomplished without much complication. Programming tools like compilers, interpreters, debuggers are used to generate the code. Different high level programming languages like C, C++, Visual basic, and Java are used for coding. With respect to the type of application, the right programming language is chosen. Once the independent programs are implemented they are linked to form the modular structure of the software according to the interface relations defined in the design document.

Activity 4: Testing and Integration

Once the code is generated, the software testing begins. Testing is the key method for dynamic verification and validation of a system. The objectives of the testing phase are to uncover and remove as many faults as possible with a minimum cost, to demonstrate the presence of all specified functionalities, and to predict the operational reliability of the product. Testing is generally focused on two areas: internal efficiency and external effectiveness. The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented. The goal of external external testing is to verify that the software is functioning according to system design and that it is performing all necessary functions or sub-functions. Initially testing begins with unit testing of independent modules

then the modules are integrated and system testing is performed followed by acceptance testing.

Activity 5: Operation and Maintenance

The system or system modifications are installed and made operational in the operational environment. The phase is initiated after the system has been tested and accepted by the user. Installation also involves user training primarily based on major system functions and support. The users are also provided installation and operation manuals. This phase continues until the system is operating in accordance with the defined user requirements. Inevitably the system will need maintenance during its operational use. During this period the software is maintained by the developer to conquer the faults that remain in it at its release time. Software will definitely undergo change once it is delivered to the customer. There are many reasons for a potential change. Change could happen because of some unexpected input values into the system. Changes in the system could directly affect the software operation. The software should be developed to accommodate changes that could happen during the post-implementation period.

The waterfall model maintains that one should move to a phase only when its proceeding phase is completed and perfected. Phases of development in the waterfall model are discrete, and there is no jumping back and forth or overlap between them. Several modifications of waterfall model are known in the literature to allow the prototyping such as phased, evolutionary, and agile development of the software. The basic difference between waterfall model and its modifications is the flexibility in the sense that the task performed in any stage of the development can be verified and validated with the previous stages so as to reduce the development cost, time, and rework. For example the user and the requirement analyst can review the specifications once they have been defined to insure that the proposed product is what the users want. This allows the user and the software team to visualize the actions performed and to find the aspects of further improvements in the accomplished tasks. Figure 1.3 demonstrates a modified waterfall model that includes reviews and feedbacks in between various development stages.

1.3 Why Software Testing Is Important

Despite using the best engineering methods and tools during each stage of the software development the software is subject to testing in order to *verify* and *validate* it (software V&V). The previous discussion on the importance of computing systems and human dependence on them clarifies the need of software testing. Bugs if appear during software operation in user environment can be fatal to the users in terms of loss of time, money, and even lives depending on criticality

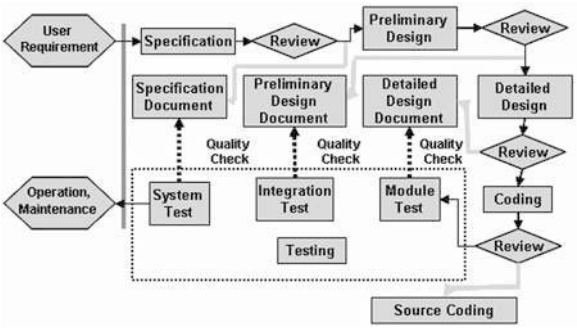


Fig. 1.3 Modified waterfall model

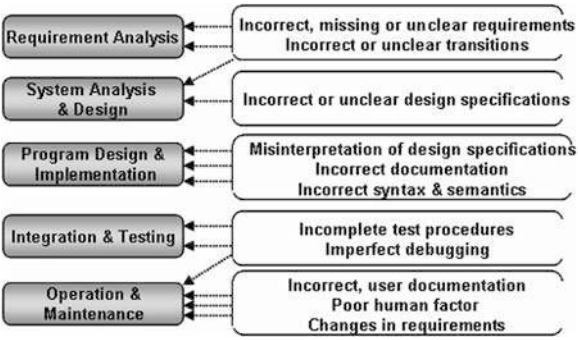


Fig. 1.4 Sources of faults in each phase of SDLC

of the function as well as to the developers in terms of cost of debugging, risk cost of failure, and goodwill loss. The bugs in the software can be manifested in each stage of its development. Figure 1.4 shows factors contributing to bugs manifestation in the various stages of SDLC.

The aim of software testing is nothing other than *quality improvement* in order to achieve the necessary reliability. Although defined in various ways basically software quality is defined as the attribute measuring how well the software product meets the stated user functions and requirements. Table 1.1 illustrates the standardized desired quality characteristics stated by ISO:IEC. Software testing involves checking processes such as inspections and reviews at each stage of the SDLC start from the requirement specification to coding. Ideally the test cases that are executed on the software to test the software are designed throughout its development life cycle. Testing is inherent to every phase of the SDLC but the testing performed in the testing stage gives confidence to developers and users on the software quality. Software testing in the testing phase is a three-stage process in which first the systems individual components, programs, and modules are tested called *unit testing*, followed by *integration testing* at subsystem and system level which includes top-down and bottom-up testing, interface testing, and stress testing and conclude with the *acceptance testing*. Figure 1.5 summarizes the different testing levels and their focus.

Table 1.1 Software quality characteristics

Functionality (Exterior quality)	Engineering (Interior quality)	Adaptability (Future quality)
Correctness	Efficiency	Flexibility
Reliability	Testability	Reusability
Usability	Documentation	Maintainability
Integrity	Structure	Compatibility

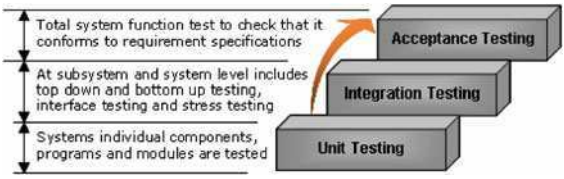


Fig. 1.5 Software testing levels

There is plethora of testing methods and testing techniques which can serve multiple purposes in different phases of SDLC. Testing is basically of four types: *defect testing, performance testing, security testing*, and *statistical testing* [7].

Defect testing is intended to find the inconsistencies between a program and its specification in contrast to validation testing that requires the system to perform correctly using an acceptance test case. A defect test case is considered to be successful if it shows the presence, not the absence, of a fault in the software. Defect testing can be performed in many different ways, the number and types of the method adopted depend on the quality requirement, software size and functionality, etc. Some of the well-known techniques are:

- *Black box testing*: is a testing method that emphasizes on executing the system functions using the input data derived from the specification document regardless to the program structure, also known as functional testing. The system functionality is determined from observing the output only; hence the tester treats the software as a black box.
- *White box testing*: Contrary to the black box testing, software is viewed as a white box or a glass box since the tests are derived from the knowledge of the software's structure and implementation hence also known as structural testing. Analysis of code can determine the approximate number of tests needed to execute all statements at least once.
- *Equivalence partitioning*: Is based on identifying equivalence partitions of the input/output data and designing the test cases so that the inputs and outputs lie within these partitions.
- *Path testing*: Here the objective is to exercise every independent path of a program with the test cases.

There is no clear boundary between these testing approaches, which can be combined during testing.

Performance testing has always been a great concern and a driving force of computer evolution which includes: resource usage, throughput, stimulus–response

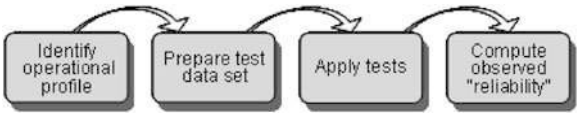


Fig. 1.6 The reliability measurement model

time, and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be considered include network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage. The goal of performance testing is performance bottleneck identification, comparison, evaluation, etc. The typical method of doing performance testing is using a benchmark program, workload, or trace designed to be representative of the typical system usage.

Security testing has become a matter of prime concern to the software developer with the detonation of worldwide web in the IT. Software security is now an attribute of software comparable to the software quality. Most of the critical and confidential software applications and services have integrated security measures against malicious attacks. The purpose of security testing for these systems include identifying and removing software flaws that may potentially lead to security violations and validating the effectiveness of security measures.

Statistical testing in contrast to other testing methods, aims to measure software reliability rather than discovering faults. It is an effective sampling method to assess system reliability and hence also known as reliability testing. Figure 1.6 illustrates the four stages of reliability assessment. Data collected from other test methods are used here to predict the reliability level achieved and which can further be used to depict the time when the desired level of quality in terms of reliability can be achieved. Reliability assessment is of undue importance to both the developers and user; it provides a quantitative measure of the number of remaining faults, failure intensity, and a number of decisions related to the development, release, and maintenance of the software in the operational phase. To the users it provides a measure for having confidence in the software quality and their level of acceptability.

1.4 Software Reliability Modeling

Previous discussion on statistical testing highlights the importance of reliability assessment in the software testing. Most of the testing methods aim to uncover the faults lying in the software. When a fault is exposed, the corresponding fault is repaired. This task of failure observation and fault removal gives an indication of improved system reliability. One of the most important things here is to know how much improvement or decline (in the case of error generation) in quality has been made. Knowledge of this information is necessary to make a quantitative measure of the software quality. Software reliability assessment during the different phases of the software development is an attractive approach to the developer as it

provides a quantitative measure of what is most important to them *software quality*. Reliability being the most dynamic software quality characteristic is preferred by the users as well as developers. As stated earlier the task of statistical testing is to measure software reliability and is performed following a set of sequential steps (Fig. 1.5). Now the question arises how we can measure the observed system reliability. Now comes the role of software reliability modeling, a sub-field of SRE. The reliability models known as Software Reliability Growth Models (SRGM) can be used here to estimate the current level of reliability achieved and to predict the time when the desired system reliability can be achieved. However, computing an appropriate measure of reliability is difficult [7]; it is associated with many difficulties such as:

- *Operational profile uncertainty*: It is difficult to simulate operational profile, which is a reflection of the real user operational profile accurately.
- *High costs of test data generation*: Defining the large set of test data that covers each program statement, path, functions, etc. is very costly as it requires long time, expert experience.
- *Statistically uncertainty*: Statistically significant number of failure data is required to allow accurate reliability measurement; measurements made with insufficient data involve huge uncertainty. With this choice of the appropriate metric and model used, add to the uncertainty of the reliability measurement.

Despite all these challenges to reliability measurement, reliability of the software is assessed during the different phases of the software development and is used for practical decision making. Before we discuss how the reliability measure is actually made we must clearly understand the difference between the software failures, faults, and errors [2, 3]. A *software failure* is a software functional imperfection resulting from the occurrence(s) of defects or faults. A *software fault* or bug is an unintentional software condition that causes a functional unit to fail to perform its required function due to some error made by the software designers. An *error* is a mistake that unintentionally deviates from what is correct, right, or true; a human action that results in the creation of a defect or fault.

Reliability assessment typically involves two basic activities—*reliability estimation* and *reliability prediction*. Estimation activity is usually retrospective and determines achieved reliability from a point in the past to present using the failure data obtained during system test or operation. The prediction activity usually involves future reliability forecast based on available software metrics and measures. Depending on the software development stage this activity involves either early reliability prediction using characteristics of the software and software development process (case when failure data are unavailable) or parameterization of the reliability model used for estimation and utilizes this information for reliability prediction (case when failure data are available). In either activity, reliability models are applied on the collected information, and using statistical inference techniques, reliability assessment is carried out.

Widespread research has been carried in the literature in the field of software reliability modeling, and several stochastic models and their applications have

been developed during the past 40 years. Many eminent researchers from the fields of stochastic modeling, reliability engineering, operational research, etc. have done excellent work in this field. Reliability growth models have been developed and validated, investigating various concepts and conditions existing in the real testing environments. Several approaches have been followed for developing these models. Many attempts have been made to classify the models into different categories so as to facilitate their application for a particular case. There exist few models, which are used widely and provide good results in a number of applications. However, which model is best for a particular application is still a big question to be answered, even though many researchers have worked to explore this aspect and provided some guidelines to select best models for certain applications. Unification of models is a recent approach in this direction. Looking at this broad area of research and having strong research interests in this field, since years we conceptualize this book to bring this literature on a platform which can be used by every one who wants to get the core knowledge of the field, know the existing work in the field so as to do the further enhancement and use it for practical application. Now we briefly discuss some preliminary concepts of software reliability modeling.

1.5 Preliminary Concepts of Reliability Engineering

Origin of hardware reliability theory is a long history. It seems to be originated during the World War II. The fundamental concepts and hardware reliability models were built on the concepts of probability theory and stochastic modeling. In the view of theorists software reliability is a concept originated from hardware reliability. In this section we discuss the fundamental concepts of software reliability and other metrics associated with software reliability study. We also provide some common distribution functions and derive the reliability measures based on them. In the later sections we discuss the stochastic processes used in the reliability study, and a detailed discussion on non-homogeneous Poisson process (NHPP)-based reliability modeling is carried to provide the readers the basic concepts of NHPP-based software reliability growth modeling.

The reliability measure applied either to hardware or software is related to their quality. Hardware reliability study aims to systematic system analysis in order to reduce and eliminate the hardware failures; in contrast the software reliability aims to analyze the system reliability growth due to testing activity in the software development. This makes the basic difference between the reliability analysis of hardware and software systems. Despite this basic difference there exist several similarities and dissimilarities between hardware reliability and software reliability. First we carry out a comparison between the two, which enables us in building better understanding of software reliability modeling.

1.5.1 Let Us Compare: Software Versus Hardware Reliability

Reliability measure applied either to software or hardware refers to the quality of the product and strives systematically to reduce or eliminate system failures. The major difference in the reliability analysis of the two systems is due to their *failure mechanism* [3, 5]. Failures in hardware is primarily due to material deterioration, aging, random failures, misuse, changes in environmental factors, design errors, etc. while software failures are caused by incorrect logic, misinterpretation of requirements and specifications, design errors, inadequate and insufficient testing, incorrect input, unexpected usage, etc. Software faults are more difficult to visualize, classify, detect, and correct due to no standard techniques available for the purpose and require a thorough understanding of the system, uncertain operational environment, and testing and debugging process.

Another important difference in the reliability analysis of the two systems lies in their *failure trend*. Failure curve that is related to hardware systems is typically a bathtub curve with three phases—burn-in, useful life, and wear-out phase as shown in Fig. 1.7. Software on the other hand does not have stable reliability in the life phase instead it enjoys reliability growth or failure decay during testing and operation since software faults are detected and removed during these phases as shown in Fig. 1.8. The last phase of the software is different from the hardware in the sense that it does not wear out but becomes obsolete due to major improvement in the software functions and technology changes.

Hardware reliability theory relies on the analysis of stationary processes, because only physical defects are considered. However, with the increase of the software system size and complexity, reliability theory based on stationary process becomes unsuitable to address non-stationary phenomenon such as reliability growth. This makes software reliability a challenging problem, which requires employment of several intelligent methods to attack [1] and forms a basis for software engineering method to base on the construction of models representing the system failure and fault removal process in terms of model parameters. The difference in the affect of fault removal requires the software reliability to be defined differently from the

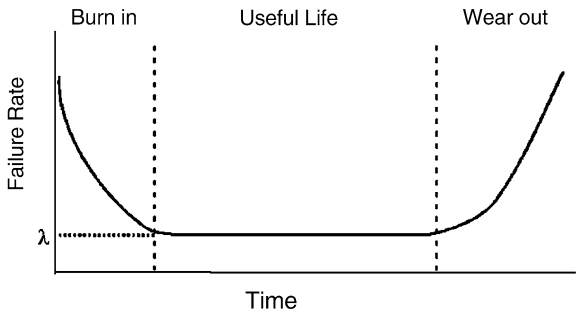


Fig. 1.7 Hardware failure curve

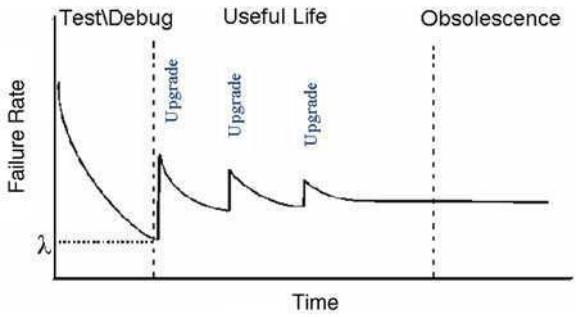


Fig. 1.8 Software failure curve

hardware reliability. On the removal of a fault, a hardware component returns to its previous level of reliability subject to the reliability of repair activity. But a software repair implies either reliability improvement (case of perfect repair) or decline (case of imperfect repair). The techniques of hardware reliability aim to maintain the hardware's standard reliability and improvements if required in the design standards. On the other hand SRE aims to continuous system reliability improvement.

1.5.2 Reliability Measures

Quantities related to the reliability measurement are most of the time defined in relation to time. In order to elaborate this point let us first define reliability. Statistically, reliability is defined as the "probability that software system will perform its function failure free under the specified environmental conditions and design limits for a specified period of time". This definition of reliability needs a careful understanding; first of all the statement that *it will perform its function* means the intended use of the software, defined in the specification and requirement documents. The *specified environmental and design limits* are defined by its software and hardware compatibility and operational profile. We cannot expect software working without failure on an input state run for which the software is not designed or working perfectly without the accurate support of supporting software and hardware. Now comes the concept of time, here we are interested in three types of time—execution time, calendar time, and clock time.

- *Execution time*: It is the processor's actual time span on an input state run, i.e., the time which is spent on running a test case of a specified run of system function.
- *Calendar time*: This type of time component is related to the number of days, weeks, months, or years the processor spends on running a system function.
- *Clock time*: This time component is related to the elapsed time from the start of a computer run to its termination. It is clear that the waiting and execution time of other programs is included in this component on a parallel processor.

Note that system down time is not included in the execution and clock time component. It is important to know that most of the software reliability models are based on the calendar time component as often the actual failure data sets are defined on the calendar time component, but nowadays the execution time component is preferred in many cases as it is accepted by most of the researchers that results are better with the execution time component. Even then we need to relate the execution time component to the calendar time as this is more meaningful to the software engineers and developers. Now we define the software reliability mathematically.

1.5.2.1 Mathematical Definition of Reliability

If the reliability of a system $R(t)$ is defined from time 0 to a time point t then it is given as

$$R(t) = P(T > t), \quad t \geq 0 \tag{1.5.1}$$

where T is a random variable denoting the time to failure or failure time of the system.

1.5.2.2 Failure Time Distribution and Reliability Measure

Consider $F(T)$, defined as the probability that if the system will fail at time t then

$$F(t) = P(T \leq t), \quad t \geq 0$$

$F(T)$ is called the failure time distribution function. Now if $f(t)$ is the density function of random variable T then we can write

$$R(t) = \int_t^\infty f(s)\,\mathrm{d}s \tag{1.5.2}$$

which is equivalent to

$$f(t) = -\frac{\mathrm{d}}{\mathrm{d}t}(R(t))$$

Further from (1.5.1) and (1.5.2) we can write

$$R(t) = 1 - F(t) \tag{1.5.3}$$

As such $F(T)$ is also called unreliability measure. On the other hand the density function can be expressed as $\lim_{\Delta t \to 0} P(t < T \leq t + \Delta t)$ meaning that the failure

time will occur between the operating time t and the next interval of operation $(t + \Delta t)$.

It is important to mention here that the reliability measure has a meaning only when it is defined with a time statement, i.e., if we say reliability of a system is 0.99, it is meaningless until it is defined on a time period. A valid statement of reliability can be "Reliability of software is 0.99 for a mission time of 4 weeks". Hence we can say that "reliability measure is a function of the mission time". A direct implication of this statement is that as the time interval on which we define the reliability measure increases, the system becomes more likely to fail. As such the reliability defined over a time interval of infinite length is zero. This statement also follows from (1.5.3) as

$$F(\infty) = 1 \Rightarrow R(\infty) = 0 \tag{1.5.4}$$

1.5.2.3 System Mean Time to Failure

We define system mean time to failure (MTTF) as the *expected time during which a system or component is expected to perform successfully without failure.* Mathematically it can be defined in terms of the system failure time density function $f(t)$ as

$$\text{MTTF} = \int_0^\infty tf(t)\,\mathrm{d}t \tag{1.5.5}$$

Using the relationship between reliability and unreliability function we can define this quantity in terms of reliability function as

$$\text{MTTF} = -\int_0^\infty t\frac{\mathrm{d}}{\mathrm{d}t}R(t)\,\mathrm{d}t = -\int_0^\infty t\mathrm{d}(R(t)) = -[tR(T)]_0^\infty + \int_0^\infty R(t)\,\mathrm{d}t$$

Now as $tR(T) \to 0 \quad \text{as } t \to 0 \quad \text{or} \quad t \to \infty$ using (1.5.4), this implies

$$\text{MTTF} = \int_0^\infty R(t)\,\mathrm{d}t \tag{1.5.6}$$

MTTF is one of the most widely used reliability measure. The measure is to be used when the distribution of failure time is known as we can make out from (1.5.5) and (1.5.6) that it depends on the failure time distribution. One important point to be noted here is that it is a measure of average time of system failure and cannot be understood as the *guaranteed minimum lifetime* of the system.

1.5.2.4 Hazard Function

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

$$\int_{t_1}^{t_2} f(t)\,\mathrm{d}t = \int_{0}^{t_2} f(t)\,\mathrm{d}t - \int_{0}^{t_1} f(t)\,\mathrm{d}t$$

$$\int_{t_1}^{t_2} f(t)\,\mathrm{d}t = F(t_2) - F(t_1) \tag{1.5.7}$$

Using (1.5.4) we can rewrite (1.5.7) in terms of reliability function as

$$\int_{t_1}^{t_2} f(t)\,\mathrm{d}t = R(t_1) - R(t_2)$$

Now we can define the rate at which failures occur in a certain time interval $[t_1, t_2]$ *as the probability that a failure per unit time occurs in the interval, given that a failure has not occurred prior to* t_1, i.e., the failure rate is defined mathematically as

$$\frac{\int_{t_1}^{t_2} f(t)\,\mathrm{d}t}{(t_2 - t_1)R(t_1)} = \frac{R(t_1) - R(t_2)}{(t_2 - t_1)R(t_1)} \tag{1.5.8}$$

The hazard function is defined as the limit of the failure rate or it can be called as instantaneous failure rate and can be derived from (1.5.8). If we redefine length of the time interval as $[t, t + \Delta t]$, the failure rate can be defined as

$$\frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} \tag{1.5.9}$$

and hazard function $h(t)$ can be obtained taking limit $\Delta t \to 0$, hence

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

The quantity $h(t)\ \mathrm{d}t$ represents the probability that the system age will fall in the small interval of time $[t, t + \Delta t]$. The hazard function reflects the picture of failure changes over the systems' or components' life. The hazard function must satisfy two conditions:

1. $h(t) \geq 0 \quad \forall\, t \geq 0$
2. $\int_0^\infty h(t)\,\mathrm{d}t = \infty$

1.5.3 Reliability Function Defined for Some Commonly Used Distributions in Reliability Modeling

The previous discussion on the reliability measures enables us to define them for some commonly used distributions in the software reliability modeling. Below we derive the reliability and hazard functions for the various distribution functions.

1.5.3.1 Binomial Distribution

The binomial distribution is a commonly used discrete random variable distribution in reliability and quality analysis. The application of the distribution is in the situations when we are dealing with the cases where an event can be expressed by binary values, e.g., success or a failure, occurrence or non-occurrence, etc.

The binomial distribution gives the discrete probability distribution of obtaining exactly x successes out of n Bernoulli trials (where the result of each trial is true with probability p and false with probability $q = 1 - p$). The binomial distribution is, therefore, given by

$$P(X = x) = \binom{n}{x} p^x q^{n-x}; \quad x = 0, 1, 2, \ldots, n \tag{1.5.10}$$

where $\binom{n}{x} = \frac{n!}{x!(n-x)!}$ is the binomial coefficient.

When $n = 1$, the binomial distribution is a Bernoulli distribution, an event which can be expressed by binary values.

The reliability function, $R(k)$, meaning here that k out of n items are good is given by

$$R(k) = \sum_{x=k}^{n} \binom{n}{x} p^x q^{n-x} \tag{1.5.11}$$

1.5.3.2 Poisson Distribution

The Poisson distribution arises in relation to Poisson processes, applicable to various phenomena of discrete nature (that is, those that may happen 0, 1, 2, 3,… times during a given period of time or in a given area) whenever the probability of the phenomenon happening is constant in time or space. Applications of the distribution are similar to that of binomial distribution, the main difference lies in the fact the sample size n is very large and may be unknown and probability p of successes is very small. However, it is also a discrete distribution with *pdf* given as

$$P(X = x) = \frac{(\lambda t)^x \mathrm{e}^{-\lambda \mathrm{t}}}{x!}; \quad x = 0, 1, 2, \ldots, \tag{1.5.12}$$

where λ is a positive real number, equal to the expected number of occurrences that occur during the given interval. The probability $P(X = x)$ represents that there are exactly x occurrences (x being a non-negative integer, $x = 0, 1, 2,\ldots$) of the event.

The above density function is the limit of binomial pdf if we substitute $\lambda = np$ and take limit $n \to \infty$.

The reliability function, $R(k)$, the probability that k or lesser number of failures occurs by time t, is given by

$$R(k) = \sum_{x=0}^{k} \frac{(\lambda t)^x \mathrm{e}^{-\lambda \mathrm{t}}}{x!} \tag{1.5.13}$$

1.5.3.3 Exponential Distribution

Exponential distribution is a continuous time distribution used extensively in the hardware and software reliability studies. The distribution describes the lengths of the inter-arrival times in a homogeneous Poison process. The exponential distribution can be looked as a continuous counterpart of the geometric distribution, which describes the number of Bernoulli trials necessary for a discrete process to change state. Exponential distribution describes the time for a continuous process to change state.

The extensive applications of this distribution in reliability study are due to the fact that it has a constant hazard function or failure rate, which reduces the complexity of mathematics involved in analysis. However, the constant hazard function has the drawback that it is appropriate only when the state of the component any time during its operation is identical to its stage at the start of its operation. This is not true in many cases. Hence the distribution is well suited to model the constant hazard rate portion of component life cycle, and not for the over all life time. This property of exponential distribution is called memoryless

property. Before we define this property mathematically, first we write the *pdf* of the distribution.

$$f(t) = \frac{1}{\theta} e^{-\frac{t}{\theta}} = \lambda e^{-\lambda}; \quad t \geq 0 \tag{1.5.14}$$

and the reliability function is given as

$$R(t) = e^{-\frac{t}{\theta}} = e^{-\lambda}; \quad t \geq 0 \tag{1.5.15}$$

where $\theta = \frac{1}{\lambda} > 0$ is the rate parameter.

The hazard function is calculated as

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

Now we state the two important properties of the exponential distribution.

Property 1 *Memoryless property*
The distribution satisfies

$$P[T \geq t] = P[T \geq t + s | T \geq s] \quad for t > 0,\ s > 0$$

The result means that the conditional reliability function for a component's lifetime that is operating by time s starting from 0 *is identical to that of a new component. This is known as "as good as new" assumption for an old component.*

Property 2 *If $T_1, T_2, \ldots, T_n$ are independently and identically distributed exponential random variables with a constant failure rate λ then*

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

where $\aleph^2(r)$ is a chi-square distribution with degree of freedom r. This result is useful for establishing a confidence interval for λ.

1.5.3.4 Normal Distribution

Normal distribution, also called the Gaussian distribution, is important continuous probability distributions. The distribution is defined by two parameters, location and scale: the mean ("average", μ) and variance (standard deviation squared, σ^2), respectively. The pdf is given by

$$f(t) = \frac{1}{\sigma\sqrt{2\prod}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}; \quad -\infty < t < \infty \tag{1.5.16}$$

and the reliability function is given as

$$R(t) = \int_{t}^{\infty} \frac{1}{\sigma\sqrt{2\prod}} e^{-\frac{1}{2}\left(\frac{s-\mu}{\sigma}\right)^2} ds \tag{1.5.17}$$

A closed form solution of the reliability function is not obtainable; however, the reliability values can be determined from the standard normal density function. Tables are easily available (see Appendix A) for standard normal distribution, which can be used to find the normal probabilities.

If $Z = \frac{t-\mu}{\sigma}$ is substituted in (1.5.16) we obtain

$$f(t) = \frac{1}{\sqrt{2\prod}} e^{-z^2/2}; \quad -\infty < Z < \infty \tag{1.5.18}$$

the above density function is called standard normal pdf, with mean 0 and variance 1. The standard normal cdf is given by

$$\Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\prod}} e^{-s^2/2} ds \tag{1.5.19}$$

Hence if T is a normal variable with mean μ and standard deviation σ then,

$$P[T \le t] = P\left[Z \le \frac{t-\mu}{\sigma}\right] = \Phi[(t-\mu)/\sigma] \tag{1.5.20}$$

and the value of $\Phi[(t-\mu)/\sigma]$ can be obtained from the standard normal table.

The normal distribution takes the well-known bell shape and is symmetrical about the mean whereas the spread is measured by the variance. The importance of the normal distribution as a model of quantitative phenomena in the natural and behavioral sciences is due in part to the central limit theorem. Many measurements ranging from psychological to physical phenomena can be approximated, to varying degrees, by the normal distribution.

The hazard function of the normal distribution given as

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

$$h(t) = \left(e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}\right) \Bigg/ \left(\int_{t}^{\infty} e^{-\frac{1}{2}\left(\frac{s-\mu}{\sigma}\right)^2} ds\right)$$

is a monotonically increasing function of time t, as

$$h'(t) = \frac{R(t)f'(t) + f^{2'}(t)}{R^2(t)} \ge 0 \quad \forall \quad -\infty < t < \infty$$

1.5.3.5 Weibull Distribution

The Weibull distribution is one of the most widely used lifetime distributions in reliability engineering. It is a versatile distribution in that it can take on the characteristics of other types of distributions, based on the value of the shape parameter, β. As said previously that exponential distribution although used more often in reliability modeling, suffers from the drawback that hazard function is constant over the components life. The Weibull distribution on the other hand can be called as a generalization of the exponential distribution due to its versatile nature.

The Weibull probability density can be given by a three-, two-, or a one-parameter function. The three-parameter Weibull pdf is given by

$$f(t) = \frac{\beta}{\theta}\left(\frac{t-\gamma}{\theta}\right)^{\beta-1} e^{-\left(\frac{t-\gamma}{\theta}\right)^{\beta}} \tag{1.5.21}$$

where $t, f(t) \geq 0$ or $\gamma, \beta, \theta > 0$, $-\infty < \gamma < \infty$and θ is scale parameter, β is shape parameter (or slope), and γ is location parameter.

The two-parameter Weibull pdf is obtained by setting $\gamma = 0$ and is given by

$$f(t) = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1} e^{-\left(\frac{t}{\theta}\right)^{\beta}} \tag{1.5.22}$$

and the one-parameter Weibull pdf is obtained by again setting $\gamma = 0$ and assuming $\beta = C$ (a constant or an assumed value)

$$f(t) = \frac{C}{\theta}\left(\frac{t}{\theta}\right)^{C-1} e^{-\left(\frac{t}{\theta}\right)^{C}} \tag{1.5.23}$$

where the only unknown parameter is the scale parameter, θ. Note that in the formulation of the one-parameter Weibull, we assume that the shape parameter β is known a priori from past experience on identical or similar products. The advantage of doing this is that data sets with few or no failures can be analyzed.

The three-parameter Weibull cumulative density function, cdf, is given by

$$F(t) = 1 - e^{-\left(\frac{t-\gamma}{\theta}\right)^{\beta}}$$

The reliability function for three-parameter Weibull distribution is hence given by

$$R(t) = 1 - e^{-\left(\frac{t-\gamma}{\theta}\right)^{\beta}} \tag{1.5.24}$$

The Weibull failure rate function, $h(t)$, is given by

$$h(t) = \frac{f(t)}{R(t)} = \frac{\beta}{\theta}\left(\frac{t-\gamma}{\theta}\right)^{\beta-1}$$

It can be shown that the hazard function is decreasing for $\beta < 1$and increasing for $\beta > 1$, and constant for $\beta = 1$. Depending on the values of the parameters, the Weibull distribution can be used to model a variety of life behaviors.

Rayleigh and exponential distributions are special cases of Weibull distribution at $\beta = 2$, $\gamma = 0$ and $\beta = 1$, $\gamma = 0$, respectively.

1.5.3.6 Gamma Distribution

The Gamma distribution is widely used in engineering, science, and business to model continuous variables that are always positive and have skewed distributions. The gamma distribution is a two-parameter continuous probability distribution. The failure density function for gamma distribution is

$$f(t) = \frac{1}{\Gamma(\alpha)\beta}\left(\frac{t}{\beta}\right)^{\alpha-1} \mathrm{e}^{-\left(\frac{t}{\beta}\right)}; \quad t \geq 0,\ \alpha, \beta > 0 \tag{1.5.25}$$

where α, β are the shape and scale parameters, respectively. The scale parameter β has the effect of stretching or compressing the range of the Gamma distribution. A Gamma distribution with $\beta = 1$ is known as the standard Gamma distribution. If β is an integer, then the distribution represents the sum of β independent exponentially distributed random variables, each of which has a mean of α (which is equivalent to a rate parameter of α^{-1}). While α controls the shape of the distribution, when $\alpha < 1$, the Gamma distribution is exponentially shaped and asymptotic to both the vertical and horizontal axes, for $\alpha = 1$ and scale parameter b gamma distribution is the same as an exponential distribution of scale parameter (or mean) b. When α is greater than one, the Gamma distribution assumes a unimodal, but skewed shape. The skewness reduces as the value of α increases.

The reliability function is given as

$$R(t) = \int_t^{\infty} \frac{1}{\Gamma(\alpha)\beta}\left(\frac{s}{\beta}\right)^{\alpha-1} \mathrm{e}^{-\left(\frac{s}{\beta}\right)}\, \mathrm{d}s \tag{1.5.26}$$

The gamma distribution is most often used to describe the distribution of the amount of time until the nth occurrence of an event in a Poisson process, i.e., when the underlying distribution is exponential. For example, customer service or machine repair. Thus if X_i is exponentially distributed with parameters $\theta = 1/\beta$, then $T = X_1 + X_2 + \cdots + X_n$ is gamma distributed with parameters β and n.

1.5.3.7 Beta Distribution

Beta distribution is a continuous distribution function defined on the interval [0, 1] parameterized by two positive shape parameters, typically denoted by α and β.

The beta distribution is used as a prior distribution for binomial proportion in Bayesian analysis. The probability density is given as

$$f(t) = \frac{t^{\alpha-1}(1-t)^{\beta-1}}{\mathbf{B}(\alpha,\beta)}; \quad 0<t<1,\ \alpha,\beta>0 \tag{1.5.27}$$

Here $\mathbf{B}(\alpha, \beta)$ is the beta function $\mathbf{B}(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$.

The reliability function is given as

$$R(t) = \int_t^{\infty} \frac{s^{\alpha-1}(1-s)^{\beta-1}}{\mathbf{B}(\alpha,\beta)}\,\mathrm{d}s \tag{1.5.28}$$

1.5.3.8 Logistic Distribution

The logistic distribution is a continuous probability distribution whose cumulative distribution function has the form of logistic function. The logistic distribution and the S-shaped pattern that results from it have been extensively used in many different areas. It is used widely in the field of reliability modeling, especially software reliability. The distribution is often seen in logistic regression and feed forward neural networks. It resembles the normal distribution in shape but has heavier tails (higher kurtosis). It is a two-parameter distribution function whose pdf is given as

$$f(t) = \frac{b(1+\beta)\mathrm{e}^{-bt}}{(1+\beta\mathrm{e}^{-bt})^2}; \quad t\geq 0,\ 0\leq b\leq 1,\ \beta\geq 0 \tag{1.5.29}$$

The cumulative density function is given as

$$F(t) = \int_0^t f(s)\,\mathrm{d}s = \frac{1-\mathrm{e}^{-bt}}{1+\beta\mathrm{e}^{-bt}} \tag{1.5.30}$$

The reliability function of the distribution can be obtained from (1.5.30) using

$$R(t) = 1 - F(t) = \frac{(1+\beta)\mathrm{e}^{-bt}}{1+\beta\mathrm{e}^{-bt}}$$

Another type of logistic distribution known as half logistic distribution can be defined, which is a one-parameter continuous probability distribution; the pdf is given as

$$f(t) = \frac{2b\mathrm{e}^{-2bt}}{(1+\mathrm{e}^{-bt})^2}; \quad t\geq 0,\ 0\leq b\leq 1,$$

and the cdf is given as

$$F(t) = \int_0^t f(s)\,\mathrm{d}s = \frac{1 - \mathrm{e}^{-bt}}{1 + \mathrm{e}^{-bt}} \tag{1.5.31}$$

In this section we have discussed the various distributions commonly used in the reliability analysis of software systems. The literature of stochastic models in reliability study of software systems is pretty wide. Knowing which model is *best* for any particular real application is very difficult. It necessitates classification of existing models into different categories according the various existing and potential future applications and formulates some guidelines for selection of *best* models in a specific situation. In the next section we put on some discussions on software reliability model classification and model selection.

1.5.4 Software Reliability Model Classification and Selection

1.5.4.1 Model Classification

Reliability models are powerful tools of SRE for estimating, predicting, controlling, and assessing software reliability. A software reliability model specifies the general form of dependence of the failure process/reliability metrics and measurements on some of the principle factors that affect it: software and development process characteristics, fault introduction, fault removal, testing efficiency and resources, and the operational environment. Software reliability modeling has been a topic of practical and academic interest since the 1970s. Today the number of existing models exceeds hundred with more models developing every year. It is important to classify the existing models in the literature into different categories so as to simplify the model selection by the practitioners and further enhancement of the field. There have been various attempts in the literature to classify the existing models according to various criteria. Goel [8] classified reliability models into four categories, namely, time between failure models, error count models, error seeding models, and input domain models. Classification due to Musa et al. [9] is according to time domain, category, and the type of probabilistic failure distribution. Some other classifications are given by Ramamoorthy and Bastani [10], Xie [11], Popstojanova and Trivedi [12]. A recent study due to Asad et al. [13] classified software reliability models according to their application to the phases of SDLC into six categories. The proposed classification of software reliability models according to phases of SDLC is shown in Fig. 1.9 along with the names of some known models from each category.

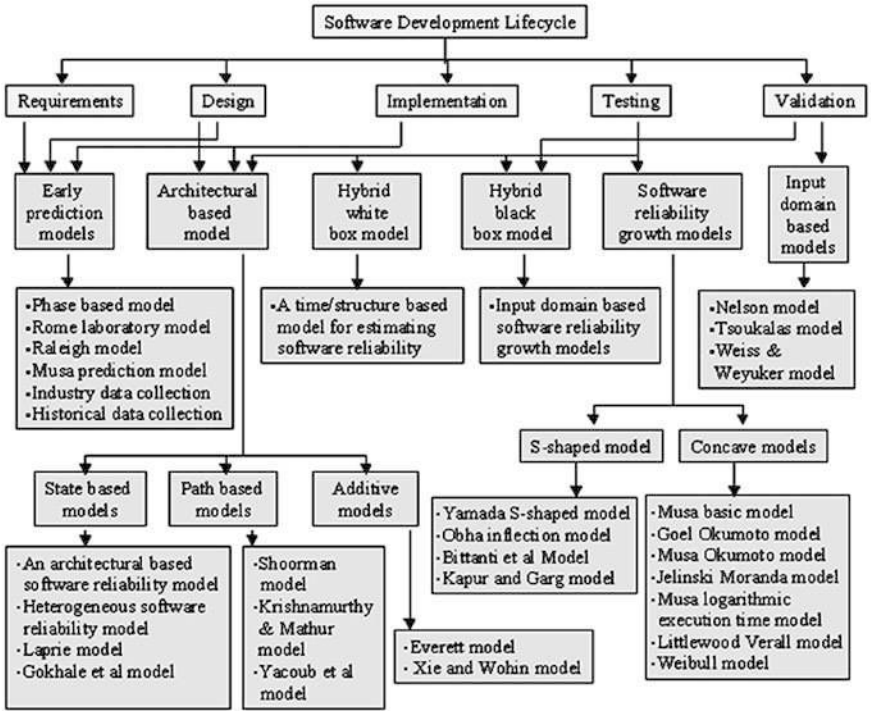


Fig. 1.9 Model classification

Early Prediction Models

These types of models use characteristics of the software development process from requirements to test and extrapolate this information to predict the behavior of software during operation.

Architecture-Based Models

These models put emphasis on the architecture of the software and derive reliability estimates by combining estimates obtained for the different modules of the software. The architecture-based software reliability models are further classified into *State-based models*; *Path-based models*; and *Additive models*.

Software Reliability Growth Models

These types of models capture failure behavior of software during testing and extrapolates it to determine its behavior during operation using failure data

information and observed trends deriving reliability predictions. The SRGM are further classified as *Concave models* and *S-shaped models*.

Input Domain-Based Models

These models use properties of the input domain of the software to derive a correctness probability estimate from test cases that executed properly.

Hybrid Black Box Models

These models combine the features of input domain-based models and SRGM.

Hybrid White Box Models

The models use selected features from both white box models and black box models. However, since the models consider the architecture of the system for reliability prediction, these models are considered as hybrid white box models.

The early prediction and architecture based models are together known as called as *white box models* which regard software as consisting of multiple components, while software reliability growth models and input domain based models are together known as *black box models* which regard software as a single unit. Black box models are studied widely by many eminent research scholars and engineers. Popstojanova and Trivedi [12] classified black box models as *failure rate models, failure count models, static models, Bayesian models, and Markov models*. Most of the research work in software reliability modeling is done on failure count models, Bayesian models, and Markov models. We give below a brief description of these categories.

Fault counting models: A fault counting model describes the number of times software fails in a specified time interval. Models in this category are assumed to describe the failure phenomenon by stochastic processes in discrete and continuous time space like homogeneous Poisson process (HPP), NHPP, compound Poisson process, etc. The majority of these failure count models are based upon the NHPP. The pioneering attempt in NHPP-based software reliability has been made by Goel and Okumoto [14]. The content of the book focuses on the development and application of NHPP based software reliability growth models. Detailed discussion on these models is carried in the Sect. 1.5.6 of this chapter.

Markovian models: A Markov process represents the number of detected faults in the software system by a Markov process. The state of the process at time t is the number of faults remaining at that time. If the fault removal process is perfect it is represented by a pure death Markov model. If the fault removal is imperfect, i.e., new faults could be introduced while debugging, then the model is represented by a birth–death Markov process. A Markov process is characterized by its state space

together with the transition probabilities between these states. The Markov assumption implies the memoryless property of the process, which is a helpful simplification of many stochastic processes and is associated with the exponential property. Jelinski and Moranda (JM) [15], Schick and Wolverton [16], Cheung [17], Goel [8], Littlewood [18] are examples of some Markov models. JM model was the earliest in this category and the basis of future Markov models.

Models based on Bayesian analysis: In the previous two categories the unknown parameters of the models are estimated either by the least squares method or by the maximum likelihood method (later in this chapter both these methods are briefly discussed). But in this category of models, the Bayesian analysis technique is used to estimate the unknown parameters of the models. This technique facilitates the use of information obtained by developing similar software projects. Based on this information the parameters of given model are assumed to follow some distribution (known as priori distribution). Given the software test data and based on a priori distribution, a posterior distribution can be obtained which in turn describes the failure phenomenon. Littlewood and Verrall [19] proposed the first software reliability model based on Bayesian analysis. Littlewood and Sofer [20] presented the Bayesian modification of JM model; Singpurwalla [21] and Singpurwalla and Wilson [22] have proposed a number of Bayesian software reliability models for different testing environments.

1.5.4.2 Model Selection

A very important aspect of software reliability modeling and application of the models to the reliability measurement is to determine which model should be used for a particular situation. Models that are good in general are not always the best choice for a particular data set, and it is not possible to know in advance which model should be used in any particular case. We do not have a guideline with high confidence level, which can be followed to choose any particular model. No one has succeeded in identifying a priori the characteristics of software that will insure that a particular model can be trusted for reliability predictions [13]. Previously most of the tools and techniques used *trend exhibited by the data* criterion for model selection. Among the tools that rank models is AT&T SRE toolkit. This tool can be used for only few SRGM. Asad et al. [13] discussed various criteria to be used in the order of their importance to select a model for a particular situation. Following criteria are specified.

- Life cycle phase
- Output desired by the user
- Input required by model
- Trend exhibited by the data
- Validity of assumptions according to data
- Nature of project

- Structure of project
- Test process
- Development process

The authors suggests that in order to choose the best model to apply to a particular reliability measurement situation, first select the life cycle phase and then find the existing reliability models applicable to that phase. Define deciding criteria, their order of importance, and assign weights to each criterion. For each criterion give applicability weights to each model; multiplying the criterion and applicability weights, one obtains the models with high scores, which can be used to measure the reliability for that case.

1.5.5 Counting Process

Stochastic modeling has been used to develop models to represent the real system and analyze their operation since years. There are two main types of stochastic process: *continuous* and *discrete*. Among the class of discrete process, counting process is used in reliability engineering widely to describe the occurrence of an event of time (e.g., failure, repair, etc.).

A non-negative, integer-valued stochastic process, $N(t)$, is called a counting process if $N(t)$ represents the total number of occurrences of an event in the interval of time $[0, t]$ and satisfies the following 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 time interval $[t_1, t_2]$

For example, consider the event $N(t)$ of airline ticket booking. If $N(t_1)$ is the number of tickets booked up to the time t_1 and $N(t_2) - N(t_1)$ is the number of tickets booked in the time interval $[t_1, t_2]$, such that $N(t_1) \leq N(t_2)$ then $N(t)$ is a counting process. An event occurs whenever a ticket is booked.

Poisson process is used most widely to describe a counting process in reliability engineering. NHPP has been used successfully in hardware reliability analysis to describe the reliability growth and deteriorating trends. Following the trends in hardware reliability analysis many researchers proposed and validated several NHPP-based SRGM. SRGM describe the failure occurrence and\or failure removal phenomenon with respect to time (CPU time, calendar time, or execution time or test cases as unit of time) and/or resources spent on testing and debugging during testing and operational phases of the software development.

NHPP-based SRGM are broadly classified into two categories first—*continuous time models*, which use time (CPU time, calendar time or execution time) as a unit of fault detection period and second—*discrete time models*, which adopt the number of test occasions/cases as a unit of fault detection period.

1.5.5.1 NHPP in Continuous Time Space

A counting process $(N(t), t \geq 0)$ is said to be an NHPP with mean value function $m(t)$, if it satisfies the following conditions:

1. There are no failures experienced at $t = 0$, that is, $N(0) = 0$.
2. The counting process has independent increments, i.e., for any finite collection of times $t_1 < t_2 < \cdots < t_k$, the k random variables $N(t_1)$, $N(t_2) - N(t_1)$,…, $N(t_k) - N(t_{k-1})$ are independent.
3. Pr(exactly one failure in $(t, t + \Delta t) = \lambda(t) + o(\Delta t)$.
4. Pr(two or more failures in $(t, t + \Delta t) = o(\Delta t)$

where $\lambda(t)$ is the intensity function of $N(t)$. If we let $m(t) = \int_0^t \lambda(x)\,\mathrm{d}x$ them $m(t)$ is a non-decreasing, bounded function representing the mean of number of faults removed in the time interval (0, t] [2]. It can be shown that

$$\Pr[N(t) = k] = \frac{(m(t))^k \mathrm{e}^{-m(t)}}{k!}, \quad n = 0, 1, 2, \ldots \tag{1.5.32}$$

i.e., $N(t)$ has a Poisson distribution with expected value $E[N(t)] = m(t)$ for $t > 0$. and the reliability of the software in the time interval of length x is given as

$$R(x|t) = \mathrm{e}^{-(m(t+x)-m(t))} \tag{1.5.33}$$

1.5.5.2 NHPP in Discrete Time Space

A discrete counting process [$N(n)$, $n \geq 0$], (n = 0, 1, 2, …) is said to be an NHPP with mean value function $m(n)$, if it satisfies the following two conditions:

1. No failures are experienced at $n = 0$, that is, $N(0) = 0$.
2. The counting process has independent increments, implies the number of failures experienced during $(n\text{th}, (n+1)\text{th})$ test cases is independent of the history. The state $m(n + 1)$ of the process depends only on the present state $m(n)$ and is independent of its past state $m(x)$, for $x < n$.

   For any two test cases n_i and n_j where$(0 \leq n_i \leq n_j)$, we have

$$\Pr\{N(n_j) - N(n_i) = x\} = \frac{\{m(n_j) - m(n_i)\}^x}{x!} \mathrm{e}^{-\{m(n_j)-m(n_i)\}} \tag{1.5.34}$$

x = 0, 1, 2, …

The mean value function $m(n)$ which is a non-decreasing in n represents the expected cumulative number of faults detected by n test cases. Then the NHPP model is formulated as

$$\Pr\{N(n) = x\} = \frac{\{m(n)\}^x}{x!} \mathrm{e}^{-m(n)}$$

Let $\overline{N}(n)$ denote the number of faults remaining in the system after execution of the nth test run. Then we have

$$\bar{N}(n) = N(\infty) - N(n)$$

where, $N(\infty)$ represents the total initial fault content of the software.

The expected value of $\overline{N}(n)$ is given by

$$E(n) = m(\infty) - m(n)$$

where, $m(\infty)$ represents the expected number of faults to be eventually detected.

Suppose that n_d faults have been detected by the execution of n test cases. The conditional distribution of $\overline{N}(n)$, given that $N(n) = n_d$, is given by

$$\Pr\{\bar{N}(n) = y | N(n) = n_d\} = \frac{\{E(n)\}^y}{y!} e^{-E(n)} \tag{1.5.35}$$

and the probability of no faults detected between the nth and $(n + h)$th test cases, given that n_d faults have been detected by n test cases, i.e., software reliability, is given by

$$R(h/n) = \exp(-\{m(n+h) - m(n)\}) \tag{1.5.36}$$

1.5.6 NHPP Based Software Reliability Growth Modeling

NHPP-based SRGM are either *concave* or *S-shaped* depending upon the shape of the failure curve described by them. Concave models describe an exponential failure curve while second category of models describes an S-shaped failure curve [2, 3]. The two types of failure growth curves are shown in Figs. 1.10 and 1.11. The most important property of these models is that they have the same asymptotic behavior in the sense that the fault detection rate decreases as the number of detected defects increases and approaches a finite value asymptotically. The S-shaped curve

Fig. 1.10 Exponential failure curve

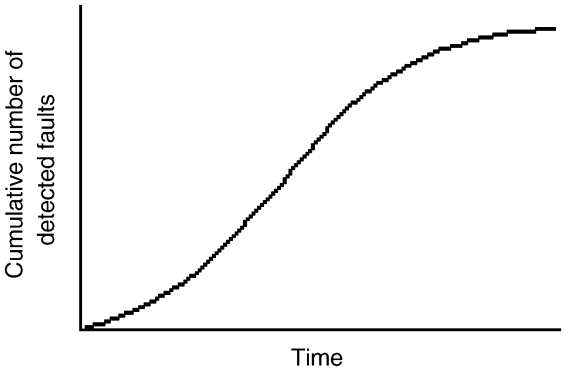


Fig. 1.11 S-shaped failure curve

describes the early testing process to be less efficient as compared to the later testing, i.e., it depicts the learning phenomenon observed during testing and debugging process.

During the last three decades several researchers devoted their research interest to NHPP-based software reliability modeling and contributed significantly in understanding the testing and debugging process and developing quality software. The primary factors analyzed and incorporated in the reliability modeling for software systems are software development process, fault tolerance, operational environment, fault removal process, testing efficiency, resources and coverage, fault severity and Error generation (Fig. 1.12).

Schneidewind [23] made the preliminary attempt in NHPP-based software reliability modeling. He assumed exponentially decaying failure intensity and rate of fault correction proportional to the number of faults to be corrected. Goel and Okumoto [14] presented a reliability model (GO model); assuming hazard rate is proportional to the remaining number of faults in the software. This research paper was a pioneering attempt in the field of software reliability growth modeling and paved the way for research on NHPP-based software reliability modeling. The model describes the failure occurrence phenomenon by an exponential curve.

The research following GO model was mainly modifying the existing research in the way of incorporating the various aspects of the real testing environment and strategy. Most of the existing NHPP-based SRGM can be categorized as follows [24]

- Modeling under perfect debugging environment
- Modeling the imperfect debugging and error generation phenomenon
- Modeling with testing effort
- Testing domain dependent software reliability modeling
- Modeling with respect to testing coverage
- Modeling the severity of faults
- Incorporating change point analysis
- Software reliability modeling for distributed software systems
- Modeling Fault detection and correction with time lag

Fig. 1.12 The primary factors analyzed and incorporated in software reliability modeling

- Managing reliability in operational phase
- Reliability Analysis of Fault Tolerant Systems
- Software reliability assessment using SDE model
- Neural network based software reliability modeling
- Discrete SRGM
- Unification of SRGM

Among the various categories mentioned above the SDE models [25–27], neural network based SRGM [28–31], unification methodologies [32–37], reliability growth analysis for fault tolerant software [38, 39] are the emerging areas and are of interest to most of the researchers. Throughout this book we will discuss several NHPP-based models developed and validated in the literature. We will also show some Operational Research applications based on these models in the later chapters of this book with numerical illustrations.

1.6 Parameter Estimation

The task of mathematical model building is incomplete until the unknown parameter of the model parameters are estimated and validated on actual software failure data sets. After selecting a model for any application, the next step is estimation of the unknown parameters of the model. In general, this is accomplished by solving an optimization problem in which the objective function (the function being minimized or maximized) relates the response variable and the functional part of the model containing the unknown parameters in a way that will produce parameter estimates that will be close to the true, unknown parameter values. The unknown parameters are, treated as variables to be solved for in the optimization, and the data serve as known coefficients of the objective function in

this stage of the modeling process. In parameter estimation one can perform either point estimation or interval estimation or both for the unknown parameters.

1.6.1 Point Estimation

In statistics, the theory of point estimation deals with use of sample data to calculate a value for the unknown parameters of the model, which can be, said a "best guess". In the statistical terms the best guess mean here that the estimated value of the parameter satisfies the following properties:

- Unbiasedness
- Consistency
- Efficiency
- Sufficiency

The theory of point estimation assumes that the underlying population distribution is known and the parameters of the distribution are to be estimated from the collected failure data. Collected failure data is either based on the actual observations from the population sample or in case the data is not available it is either collected from a similar application (population) or simulated from the developed model.

Example Assume n independent samples from the exponential density

$$f(x,\lambda) = \lambda e^{-\lambda x}; \quad x > 0,\ \lambda > 0$$

The joint pdf of the sample observations is given by

$$f(x_1,\lambda)\cdot f(x_2,\lambda)\cdots f(x_n,\lambda) = \lambda^n e^{-\lambda\sum_{i=1}^{n} x_i}; \quad x_i > 0,\ \lambda > 0$$

Now the problem of point estimation is to find a function $h(X_1, X_2, \ldots, X_n)$ such that if $x_1, x_2, \ldots, x_n$ are the observed sample values $X_1, X_2, \ldots, X_n$ then $\widehat{\lambda} = h(x_1, x_2, \ldots, x_n)$ is a good estimate of λ.

1.6.1.1 Some Definitions

Unbiased estimator: For a given positive integer n, the statistic $Y = h(X_1, X_2, \ldots, X_n)$ is called an unbiased estimator of the parameter θ if the expectation of Y is equal to the parameter θ, that is

$$E(Y) = \theta$$

Consistent estimator: The statistic Y is called a consistent estimator of the parameter θ if Y converges stochastically to a parameter θ as n approaches infinity. Where n is the sample size. If $\in$ is an arbitrarily small positive number when Y is consistent, then

$$\lim_{n\to\infty} P(|Y-\theta| \leq \in) = 1$$

Efficient estimator: The statistic Y will be called the minimum variance unbiased estimator of the parameter θ if Y is unbiased and the variance of Y is less than or equal to the variance of every other unbiased estimator of θ. An estimator that has the property of minimum variance in large samples is said to be efficient.

Sufficient estimator: The statistic Y is said to be sufficient estimator for θ if the conditional distribution of X, given $Y = y$, is independent of θ.

Cramer–Rao inequality: Let $X_1, X_2, \ldots, X_n$ denote a random sample from a distribution with pdf $f(x; \theta)$ for $\theta_1 < \theta < \theta_2$, where $\theta_1$and θ_2 are known. Let $Y = h(X_1, X_2, \ldots, X_n)$ be an unbiased estimator of θ. The lower bound inequality on the variance of Y, Var(Y), is given by

$$\mathrm{Var}(\mathrm{Y}) \geq \frac{1}{nE\left\{\left[\frac{\partial \ln f(x;\theta)}{\partial\theta}\right]^2\right\}}$$

Asymptotic efficient estimator: an estimator $\widehat{\theta}$ is said to be asymptotic efficient if θ has a variance that approaches the Cramer–Rao lower bound for large n, that is

$$\lim_{n\to\infty} \mathrm{Var}\left(\sqrt{n\widehat{\theta}}\right) = \frac{1}{nE\left\{\left[\frac{\partial \ln f(x;\theta)}{\partial\theta}\right]^2\right\}}$$

Most of the NHPP-based SRGM are described by the non-linear functions. Method of *Non-linear Least Square* (NLLS) and *Maximum Likelihood Estimate* (MLE) [2, 3, 40–43] are the two widely used estimation techniques for non-linear models. Unlike traditional linear regression, which is restricted to estimating linear models, nonlinear regression (NLR) methods can estimate models with arbitrary relationships between independent and dependent variables.

1.6.1.2 Non-Linear Least Square Method

Consider a set of observed data points (t_i, y_i); $i = 1, 2, \ldots n$, where t_i is the observation time and y_i is the observed sample value. A mathematical model of the form $m(x,t)$ is fitted on this data set. The model depends on the parameters $x = \{x_i; i = 1, 2, \ldots m\}$, for some $\widehat{x}$ we can compute the residuals,

$$f_i(\widehat{x}) = y_i - m(\widehat{x}, t_i)$$

The method of least square determines the unknown parameters of the model by minimizing the sum square of these residuals between the observed responses and the fitted value by the model. Unlike linear models, the least squares minimization cannot be done with simple calculus. It has to be accomplished using iterative numerical algorithms. Now about the quality of least square estimate, it is difficult to picture exactly how good the parameter estimates are, they are, in fact, often quite good. The accuracy of the estimates can be measured based on some goodness of fit measures (discussed in the later sections).

1.6.1.3 Maximum Likelihood Estimation Method

MLE is one of the most popular and useful statistical method for fitting a mathematical model to some data, i.e., deriving the point estimates. The idea behind MLE parameter estimation is to determine the parameters that maximize the probability (likelihood) of the sample data. From a statistical point of view, this method is considered to be more robust (with some exceptions) and yields estimators with good statistical properties. The fact that this method is versatile, apply to most models and to different types of data make it more popular. In addition, it provides efficient methods for quantifying uncertainty through confidence bounds. Although the methodology for maximum likelihood estimation is simple, the implementation is mathematically intense. Using today's computer power, however, mathematical complexity is not a big obstacle.

Consider a random sample X_1, X_2, …, X_n drawn from a continuous distribution with *pdf*

$$f(x; \theta = (\theta_1, \theta_2, \ldots, \theta_k))$$

where $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ is the vector of unknown distribution parameters, k in number. Assuming that the sample observations are independent, the likelihood function $L(X; \theta)$ is the product of the pdf of the distribution of the random samples evaluated at each sample point.

$$L(X; \theta) = L(X_1, X_2, \ldots, X_n; (\theta_1, \theta_2, \ldots, \theta_k)) = \prod_{i=1}^{k} f(X_i; \theta)$$

The likelihood estimator $\widehat{\theta}$ can now be computed by maximizing $L(X; \theta)$ with respect to θ. In practice, it is often easier to maximize $\ln L(X; \theta)$ rather than $L(X; \theta)$ due to easy of computations as compared to the actual likelihood function. The estimates of $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ obtained maximizing $\ln L(X; \theta)$ maximize $L(X; \theta)$ as logarithm function is monotonic. The log likelihood function is given by

$$\ln L(X; \theta) = \sum_{i=1}^{k} \ln f(X_i; \theta)$$

In general the mechanics of obtaining MLE can be summarized as:

(a) Find the joint density function $L(X; \theta)$.
(b) Take the natural log of the density in L.
(c) Take partial derivatives of ln L with respect to each parameter.
(d) Set partial derivatives to Zero.
(e) Solve for parameters.

Now we formulate the likelihood function for the NHPP-based software reliability models.

For the interval domain data points(t_i, y_i); $i = 1, 2, \ldots n$, where t_i is the observation time and y_i is the cumulative observed sample value by the time t_i, based on the NHPP assumptions the likelihood function is defined as

$$L \equiv \prod_{i=1}^{n} \frac{[m(t_i) - m(t_{i-1})]^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} \mathrm{e}^{-\{m(t_i) - m(t_{i-1})\}}$$

If the data set is time domain the likelihood function is defined as

$$L \equiv \prod_{i=1}^{n} \lambda(t_i) e \left(- \int_0^{t_n} \lambda(x)\, \mathrm{d}x \right)$$

Both these methods have one thing in common that the nonlinear objective function (the sum square residuals in NLLS and the likelihood function in MLE) is optimized. For finding the optimal solution manually one needs to compute the first order partial differential equations corresponding to each parameter of the problem, equate them to zero and solve the resulting system of equations. In most of the cases solving this system of equations is difficult and contrary to the linear model fitting, we cannot express analytically the solution of this optimization problem. As such it requires numerical methods, a programming algorithm to implement the numerical procedures and huge computation time to solve the problem, which is not favored by the management and software engineering practitioners. Numerical procedures are followed by a number of the researchers in their research articles. One truth related to obtaining the solution of these simultaneous equations using numerical algorithm is that the estimates obtained cannot be guaranteed to be global solution. In most of the cases they converge to the local optimum solution. As an alternative method that minimizes the efforts with reduced time requirement is to use statistical software packages such as SPSS, SAS, Mathematica, etc. in which we can use the inbuilt software functions to solve these kinds of optimization problems to find the estimates of nonlinear models. This software also uses well-defined numerical algorithms to obtain the estimates. The solutions obtained by the use of inbuilt estimation modules of software also converge to local solutions in most of the cases, but by no means we can guarantee that which of the solutions, one obtained by self programmed numerical algorithm or a software module, is better. However, using an advanced version of software

that is designed on more comprehensive numerical procedures and use different procedures to obtain the estimate can provide us with a better solution. Throughout the book we have used the Statistical Package for Social Sciences (SPSS 18.0) for the estimation of unknown parameters of the models. SPSS is a comprehensive and flexible statistical analysis and data management system. It can take data from almost any type of file and use them to generate tabulated reports, charts, and plots of distributions and trends, descriptive statistics, and conduct complex statistical analysis. SPSS *Regression Models* [44, 45] enables the user to apply more sophisticated models to the data using its wide range of NLR models.

NLR and conditional nonlinear regression (CNLR) modules of SPSS have been used to estimate the unknown parameters. The modules use the iterative estimation algorithms, namely, sequential quadratic programming (SQP) [46] and Levenberg–Marquardt (LM) method [47, 48] to find the least square estimates of the parameters. Both methods starts with an initial approximation of the parameters and at each stage improve the objective value until convergence. LM method is chosen by default in SPSS NLR function, while if there are overflow/underflow errors and failure to converge; one may select the SQP method. In the other case if overflow and underflow errors appears, bounds on the parameters are set in the form of linear constraints or there may be some other constraints on the parameter values (such as sum of few parameters has to be one) then we have to select the SQP method. SQP method minimizes the sum square residuals solving a linearly constrained quadratic sub problem in each stage. Which algorithm is *best* depends on the data. If we have to specify a nonlinear model, which has different equations for different ranges of its domain (change point and fault tolerance system models) we use the CLNR function of SPSS. In CNLR we can specify a segmented model using conditional logic. To use conditional logic within a model expression or a loss function, we form the sum of a series of terms, one for each condition. Each term consists of a logical expression (in parentheses) multiplied by the expression that should result when that logical expression is true.

1.6.2 Interval Estimation

A point estimator may not (which is true in many cases) coincide with the actual value of the parameter. In this situation it is favorable to determine an interval of possible (or probable) values of an unknown population parameter. This is called confidence interval estimation of the form $[\theta_L, \theta_U]$, where θ_L is the lower bound and θ_U is the upper bound on the parameter value.

Alternatively interval estimation is supplemented with point estimation in order to make the point estimates more useful as it provide a tolerance limit of the type lower and higher values the a point estimate can take. Statistically if $[\theta_L, \theta_U]$ is interval estimates of the point estimate θ with probability $(1 - \alpha)$, then θ_L and θ_U will be called $100(1 - \alpha)\%$ confidence limits and $(1 - \alpha)$ is called the confidence coefficient.

1.6.2.1 Confidence Intervals for Normal Parameters

The distribution has two unknown parameters, the mean ("average", μ) and variance (standard deviation squared, σ^2).

1.6.2.2 Confidence Limits for the Mean μ When σ^2 is Known

We know that the statistic $Z = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}}$ follows 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 μ is given by

$$P\left[\bar{X} - Z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + Z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\right] = 1-\alpha$$

i.e.,

$$\mu_L = \bar{X} - Z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$$

and

$$\mu_U = \bar{X} + Z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$$

1.6.2.3 Confidence Limits for the Mean μ When σ^2 is Unknown

We know the sample standard error is given by

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_1 - \bar{X})^2}$$

It can be shown that the statistic

$$T = \frac{\bar{X}-\mu}{S/\sqrt{n}}$$

follows t distribution with $(n-1)$ degrees of freedom. Thus for a given sample mean and sample standard deviation, we obtain

$$P\left[|T| < t_{\alpha/2,(n-1)}\right] = (1-\alpha)$$

Hence a 100(1 − α) % confidence interval for the mean μ is given by

$$P\left[\bar{X} - t_{\alpha/2,(n-1)}\frac{S}{\sqrt{n}} < \mu < \bar{X} + t_{\alpha/2,(n-1)}\frac{S}{\sqrt{n}}\right] = 1 - \alpha$$

1.6.2.4 Confidence Limits on σ^2

Note that $n\frac{\widehat{\sigma}^2}{\sigma^2}$ has a χ^2 distribution with $(n - 1)$ degrees of freedom. Correcting for the bias in $\widehat{\sigma}^2$, $(n-1)\frac{\widehat{\sigma}^2}{\sigma^2}$ has the same distribution. Hence

$$P\left[\chi^2_{\alpha/2,(n-1)} < \frac{(n-1)S^2}{\sigma^2} < \chi^2_{(1-\alpha/2),(n-1)}\right] = 1 - \alpha$$

or

$$P\left[\frac{\sum (x_i - \bar{x})^2}{\chi^2_{(1-\alpha/2),(n-1)}} < \sigma^2 < \frac{\sum (x_i - \bar{x})^2}{\chi^2_{\alpha/2,(n-1)}}\right] = 1 - \alpha$$

Similarly for one-sided limits we can choose $\chi^2(1 - \alpha)$ or $\chi^2(\alpha)$.

Likewise we can determine confidence intervals for the parameters of the other probability distribution function.

1.7 Model Validation

1.7.1 Comparison Criteria

Once some models have been selected for an application. Their performance can be judged by their ability to fit the observed data and to predict satisfactorily the future behavior of the process (predictive validity) (Musa 1989) [50]. Many established criteria are defined in the literature to validate the *goodness of fit* of models on any particular data and choose the most appropriate one. Some of these criteria are given below.

The mean square error (MSE): The model under comparison is used to simulate the fault data, the difference between the expected values, $\hat{y}(t_i)$; $i = 1, 2, \ldots k$ and the observed values y_i is measured by MSE as follows.

$$\text{MSE} = \sum_{i=1}^{k} \frac{(\hat{y}(t_i) - y_i)^2}{k - n}$$

where k is the number of observations, n is the number of unknown parameters in the model. The lower MSE indicates less fitting error, thus better goodness of fit [2].

Coefficient of multiple determination (R^2): is defined as the ratio of the Sum of Squares (SS) resulting from the trend model to that from a constant model subtracted from 1, that is

$$R^2 = 1 - \frac{\text{residual SS}}{\text{corrected SS}}$$

R^2 measures the percentage of the total variation about the mean accounted for by the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model does not fit the data well. The larger the value, the better the model explains the variation in the data [2].

Prediction error (PE): The difference between the observed and predicted values at any instant of time i is known as PE_i. Lower the value of Prediction Error better is the goodness of fit [49].

Bias: The average of PE is known as bias. Lower the value of Bias better is the goodness of fit [49].

Variation: The standard deviation of PE is known as variation.

$$\text{Variation} = \sqrt{\left(1/N - 1\right) \sum (\text{PE}_i - \text{Bias})^2}$$

Lower the value of Variation better is the goodness of fit [49].

Root mean square prediction error: It is a measure of closeness with which a model predicts the observation.

$$\text{RMSPE} = \sqrt{\left(\text{Bias}^2 + \text{Variation}^2\right)}$$

Lower the value of Root Mean Square Prediction Error better is the goodness of fit [49].

Observed and estimated values can be plotted on time scale to obtain the goodness of fit curves.

1.7.2 Goodness of Fit Test

The reason of carry a goodness of fit test of a statistical model is to determine how well it fits a set of observations. Measures of goodness of fit typically summarize the discrepancy between observed values and the values expected under the model in question. Such measures can be used in statistical hypothesis testing, e.g., to test for normality of residuals, to test whether two samples are drawn from identical distributions or whether outcome frequencies follow a specified distribution. The two commonly used goodness of fit tests used for reliability models are χ^2

goodness of fit test and Kolmogorov–Smirnov "d" test. Both of these tests are non-parametric. The χ^2 test assume large sample normality of the observed frequency about its mean while the "d" test assumes only a continuous distribution.

1.7.2.1 Chi-Square (χ^2) Test

The statistic $\chi^2 = \sum_{i=1}^{k} \left(\frac{x_i - \mu_i}{\sigma_i}\right)^2$ is said to follow chi-squared χ^2 distribution with k degree of freedom. The steps involved in carrying the 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 the 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 call should be greater than one. 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 $S = \sum_{i=1}^{n} \frac{(f_i - F_i)^2}{F_i}$.
5. From the χ^2 tables, choose a value of χ^2 with the desired significance level and with the degree of freedom ($= n - 1 - r$), where r is the number of population parameter estimated.
6. Reject the hypothesis that the sample distribution is the same as the theoretical distribution if

$$S = \chi^2_{(1-\alpha),(n-1-r)}$$

where α is the level of significance.

1.7.2.2 The Kolmogorov–Smirnov Test (K–S Test)

The test is based on the empirical distribution function (ECDF). Since it is non-parametric, it treats individual observations directly and is applicable even in the case of very small sample size, which is usually the case with SRGM validation. Lower the value of Kolmogorov–Smirnov test better is the goodness of fit.

Let $X_1 \leq X_2 \leq \cdots \leq X_n$ denotes the ordered sample values. Define the observed distribution function, $F_n(x)$ as follows

$$F_n(X) = \begin{cases} 0 & \text{for } x \leq x_1 \\ \frac{i}{n} & \text{for } x_i \leq x \leq x_{i+1} \\ 1 & \text{for } x \geq x_n \end{cases}$$

Assume the testing hypothesis

$$\mathrm{H}_0 : F(x) = F_0(x)$$

where $F_0(x)$ is a given continuous distribution and $F(x)$ is an unknown distribution.

Let

$$\mathrm{d}_n = \sup_{-\infty < x < \infty} |F_n(x) - F_0(x)|$$

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 Ho; otherwise, we would reject it when $d_n > d_{n,\alpha}$. The value $d_{n,\alpha}$ is given in Appendix A, where n is the sample size and α is the level of significance.

1.7.3 Predictive Validity Criterion

Predictive validity is defined as the ability of the model to determine the future behavior from present and past behavior of a process. This criterion was proposed by [9]. Suppose t_k be the time, y_k is the observed value of the event during the interval $(0, t_k]$, and $\hat{y}(t_k)$ is the estimated value determined using the actually observed data up to an arbitrary time $t_e (0 < t_e \leq t_k)$, in which (t_e/t_k) denotes the process progress ratio. The difference between the predicted value $\hat{y}(t_k)$ and the reported value y_k measures the prediction fault. The ratio $\{(\hat{y}(t_k) - y_k)/y_k\}$ is called *Relative Prediction Error* (RPE). If the RPE value is negative/positive the model is said to underestimate/overestimate the future process. A value close to zero for RPE indicates more accurate prediction, thus more confidence in the model and better predictive validity. The value of RPE is said to be acceptable if it is within $\pm 10\%$ [2, 9]. A particular model can also be judged to fit to a given data if the parameter estimates are relatively stable over some particular intervals for the various truncations.

Exercises

1. What is software reliability engineering? What are its two fundamental philosophies?
2. Explain the layered approach of SRE for the improvement of software quality.
3. The waterfall model of SDLC relates the development of software to five sequential set of process and call for the execution of these stages in sequential manner in the order they are described. What problems are usually encountered if one uses the traditional waterfall models for the software development? Suggest two alternative approaches.

4. Even if software is tested for an infinite time it cannot be guaranteed that it is 100% reliable. Comment.
5. What makes reliability a key quality characteristic? Give some other measures of software quality.
6. What is statistical testing? What is the need of quantitatively measuring the level of reliability in software?
7. Unlike hardware software reliability growth function is increasing over time, even then software has a finite life cycle. Explain.
8. Explain the difference between the reliability growth curves of hardware and software.
9. What are prime causes of fault manifestation in the different stages of SDLC?
10. How black box software testing differs from white box software testing?
11. Define failure rate or hazard rate. Derive the mathematical expression of reliability in terms of the hazard rate.
12. Show that if the hazard rate is constant then the reliability function is exponential.
13. The pdf of Normal distribution is given by

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

Derive the reliability function.

14. Define a non-homogeneous Poisson process in continuous time space.
15. What are the major factors that affect the reliability growth during testing? Explain the role of each in brief.
16. What are the properties of a good estimate of unknown parameters of an SRGM?
17. Describe briefly the least square and maximum likelihood methods of parameter estimation.
18. Explain sequential quadratic programming iterative parameter estimation algorithm.
19. How do we determine the predictive validity of a reliability model?

References

1. Lyu MR (1996) Handbook of software reliability engineering. McGraw-Hill, New York ISBN 0-7-039400-8
2. Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware and software reliability. World Scientific, Singapore
3. Pham H (2006) System software reliability. Reliability engineering series. Springer, London
4. Garfinkel S (2005) History's Worst Software Bugs http://www.wired.com/software/coolapps/news/2005/11/69355?currentPage=all
5. Musa JD (1998) Software reliability engineering. McGraw-Hill, New York ISBN 0-07-913271-5

6. Pressman RS (2005) Software engineering: a practitioner's approach, 6th edn. Mc-Graw Hill Professional, NY ISBN 0-07-285318-2
7. Sommerville L (1995) Software engineering, 5th edn. Addison Wesley Longman Publishing Co., Inc, Redwood City ISBN:0-201-42765-6
8. Goel AL (1985) Software reliability models: assumptions, limitations and applicability. IEEE Trans Softw Eng SE-11:1411–1423
9. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York ISBN 0-07-044093-X
10. Ramamoorthy CV, Bastani FB (1982) Software reliability status and perspectives. IEEE Trans Reliability 37(1):88–91
11. Xie M (1990) Software reliability modeling. World Scientific Publications, Singapore
12. Popstojanova K, Trivedi K (2001) Architecture based approach to reliability assessment of software systems. Performance Evaluation 45(2):179–204
13. Asad CA, Muhammad Ullah I, Muhammad Rehman J (2004) An approach for software reliability model selection. In: Proceedings 28th annual international computer software and applications conference (COMPSAC'04), pp 534–539
14. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliability R-28(3):206–211
15. Jelinski Z, Moranda P (1972) Software reliability research. In: Freiberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 465–484
16. Schick GJ, Wolverton RW (1978) An analysis of competing software reliability models. IEEE Trans Softw Eng 4(2):104–120
17. Cheung RC (1980) A user oriented software reliability growth model. IEEE Trans Softw Eng SE-6:118–125
18. Littlewood B (ed) (1987) Software reliability: achievement and assessment. Blackwell, Oxford
19. Littlewood B, Verrall JL (1973) A Bayesian reliability growth model for computer software. Appl Stat 22:332–346
20. Littlewood B, Sofer A (1987) A Bayesian modification to the Jelinski–Moranda software reliability model. Softw Eng J 2(2):30–41
21. Singpurwalla ND (1995) The failure rate of software: does it exist. IEEE Trans Reliability 44(3):463–469
22. Singpurwalla ND, Wilson SP (1999) Statistical methods in software engineering, reliability and risk. Springer, New York
23. Schneidewind NF (1975) Analysis of error processes in computer software. Sigplan Not 10:337–346
24. Gupta A (2009) Some contributions to modeling and optimization in software reliability and marketing. Ph.D. Thesis, Department of OR, Delhi University, Delhi
25. Yamada S, Nishigaki A, Kimura M (2003) A stochastic differential equation model for software reliability assessment and its goodness of fit. Int J Reliability Appl 4(1):1–11
26. Yamada S, Tamura Y (2006) A flexible stochastic differential equation model in distributed development environment. Eur J Oper Res 168:143–152
27. Kapur PK, Singh VB, Anand S (2007) Effect of change-point on software reliability growth models using stochastic differential equation. In: 3rd International conference on reliability and safety engineering (INCRESE-2007), Udaipur, 7–19 December 2007, pp 320–333
28. Su YS, Haung CY (2007) Neural network based approaches for software reliability estimation using dynamic weighted combinational models. J Syst Softw 80:606–615
29. Kapur PK, Khatri SK, Goswami DN (2008) A generalized dynamic integrated software reliability growth model based on artificial neural network approach. Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems. Macmillan advanced research series, pp 813–838
30. Kapur PK, Khatri SK, Basirzadeh M (2008) Software reliability assessment using artificial neural network based flexible model incorporating faults of different complexity. Int J Reliability Qual Safety Eng 15(2):113–127

31. Kapur PK, Khatri SK, Yadav K (2008) An artificial neural-network based approach for developing a dynamic integrated software reliability growth model. Presented in international conference on present practices and future trends in quality and reliability, ICONQR08, 22–25 January 2008
32. Inoue S, Yamada S (2002) A software reliability growth modeling based on infinite server queuing theory. In: Proceedings 9th ISSAT international conference on reliability and quality in design, Honolulu, HI, pp 305–309
33. Dohi T, Osaki S, Trivedi KS (2004) An infinite server queuing approach for describing software reliability growth—unified modeling and estimation framework. In: Proceedings 11th Asia-Pacific software engineering conference (APSEC'04), pp 110–119
34. Kapur PK, Kumar J, Kumar R (2008) A unified modeling framework incorporating change point for measuring reliability growth during software testing. OPSEARCH J Oper Res Soc India 45(4):317–334
35. Kapur PK, Anand S, Inoue S, Yamada S (2010) A unified approach for developing software reliability growth model using infinite server queuing model. Int J Reliability Qual Safety Eng, to appear
36. Kapur PK, Pham H, Anand S, Yadav K (2011) A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans Softw Reliability, doi 10.1109/TR.2010.2103590
37. Langberg N, Singpurwalla ND (1985) Unification of some software reliability models. SIAM J Comput 6:781–790
38. Kapur PK, Gupta A, Jha PC (2007) Reliability growth modeling and optimal release policy of a n-version programming system incorporating the effect of fault removal efficiency. Int J Autom Comput Springer 4(4):369–379
39. Kapur PK, Gupta A, Gupta D, Jha PC (2008) Optimum software release policy under fuzzy environment for a n-version programming system using a discrete software reliability growth model incorporating the effect of fault removal efficiency. Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems. Macmillan advance research series, pp 803–816
40. Putsis WP (1998) Parameter variation and new product diffusion. J Forecasting 17(3–4): 231–257
41. Hardie BGS, Fader PS, Wisniewski M (1998) An empirical comparison of new product trial forecasting models. J Forecasting 17:209–229
42. Schmittlein DC, Mahajan V (1982) Maximum likelihood estimation for an innovation diffusion model of new product acceptance. Marketing Sci 1(1):57–78
43. Meade N, Islamb T (2006) Modeling and forecasting the diffusion of innovation—a 25 year review. Int J Forecast 22(3):519–545
44. Chen X, Ender P, Mitchell M, Wells C (2003) Regression with SPSS. http://www.ats.ucla.edu/stat/spss/webbooks/reg/default.htm. Accessed 13 July 2010
45. Garson GD (2009) Nonlinear regression. http://faculty.chass.ncsu.edu/garson/PA765/nonlinear.htm. Accessed 13 July 2010
46. Abramson MA, Chrissis JW (1998) Sequential quadratic programming and the ASTROS structural optimization system. Struct Optim 15:24–32
47. Gill PR, Murray W, Wright MH (1981) Practical optimization. Academic Press, London
48. Madsen K, Nielsen KB, Tingleff O (2004) IMM methods for non-linear least squares problems. Informatics and Mathematical Modeling, Technical University of Denmark
49. Pillai EEK, Nair VSS (1997) A model for software development effort and cost estimation. IEEE Trans Softw Eng 23(8):485–497
50. Musa JD, Iannino A, Okumoto K (1989) Software reliability: measurement, prediction, application. McGraw-Hill, New York. ISBN 0-07-044093-X

Chapter 2
Software Reliability Growth Models

2.1 Introduction

Studies in software reliability modeling started as early as early 1960s. The issues related to software quality quantification and reliability measurement arose even during the time when the development of computing systems started. Since in the 1960s the cost of the computing systems was very high, use was limited to few organizations, hardware design, test and maintainability was immature, the concepts of software reliability were in infancy stage as much of the studies were concerned with the productivity and quality of the hardware systems. Haugk et al. [1] presented some experimental results concerning the testing of a switching system in which software was an essential part and contained many programming errors, clerical errors, requirement changes and program improvements. yet there was no direct approach to the study of software reliability.

The first paper on software reliability appears to have been published in 1967 due to Hudson. He viewed software development as a birth and death process in which fault generation was a birth, and fault-correction was a death. The number of faults existing at any time defined the state of the process; the transition probabilities related to the birth and death functions. He confined his work to pure death processes, for reasons of mathematical tractability. He obtained Weibull distribution of intervals between failures. Data from the system test phase of one program were presented. Jelinski and Moranda's [2] work was recognized as the second major step. They assumed a hazard rate for failures that was piecewise constant (hazard rate changed at each fault-correction by a constant amount) and proportional to the number of faults remaining. Although the hazard rate changed at each fault-correction by a constant amount, but was constant between corrections. They applied maximum likelihood estimation to determine the total number of faults existing in the software and the constant of proportionality between number of faults remaining and hazard rate. They further proposed two variants of the model known as Jelinski Moranda Geometric model and Moranda Geometric

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_2,
© Springer-Verlag London Limited 2011

Poisson model [3]. In the first model hazard rate decreases in steps that form a geometric progression. The second model has a hazard rate, which also decreases in a geometric progression, but the decrements occur at fixed intervals rather than at each failure correction. The period of seventies was most significant in the software reliability study, some models [2, 4–9] developed in this decade later formed basis of further research in the field and found many practical applications.

Shooman [9] introduced some new concepts. He viewed the hazard rate can be determined by the rate at which execution of the program resulted in the remaining faults being passed. Thus the hazard rate depended on the instruction processing rate, the number of instructions in the program and the number of faults remaining in the program. The number of remaining faults, of course, depended on the number of faults corrected, and the profile of the latter quantity as a function of time was assumed to be related to the project personnel profile in time. Several different fault-correction profiles were proposed; the choice would depend on the particular project one was working with. Schneidewind [7] investigated different reliability functions such as the exponential, normal, gamma, and Weibull for estimating software reliability and suggested to choose the distribution that best fit the particular project in question. He indicated the importance of determining confidence intervals for the parameters estimated rather than just relying on point estimates. He also suggested that the time lag between failure detection and correction be determined from actual data and used to correct the time scale in forecasts. Another, early model was proposed by Schick and Wolverton [10]. They assumed hazard rate to be proportional to the product of the number of faults remaining and the time spent in debugging. The amount of debugging time between failures has a Rayleigh distribution.

In early 1975, Musa proposed an execution time model of software reliability. This model was later studied and applied on many real software applications by many researchers and yield good results for most of them. Musa postulated that execution time is the best practical measure of failure inducing stress that was being placed on the program. Most calendar time models do not account for varying usage of the program in either test or operation. Musa considered execution time in two respects; the operating time of a product delivered to the field, and the cumulative execution time that had occurred during test phases of the development process and during post delivery maintenance. The hazard rate was assumed to be constant with respect to operating time but would change as a function of the faults remaining and hence the cumulative execution time. Use of two kinds of time separates fault repair and growth phenomena from program operation phenomena. Musa assumed that the fault-correction rate was proportional to the fault-detection or hazard rate, making consideration of debugging personnel profiles unnecessary. A calendar time component was developed for the model that related execution time to calendar time, allowing execution time predictions to be converted to dates. The calendar time component is based on a model of the debugging process.

Littlewood and Verrall [5] worked out a software reliability model based on Bayesian approach. The idea was to measure of strength of belief that a program will operate successfully rather than the outcome of an experiment to determine the number of times a program would operate successfully. Littlewood modeled the hazard rate as a random process for the failures experienced. Goel and Okumoto [4] (GO model) proposed a reliability growth model, which describes the failure detection, as a non-homogeneous Poisson process (NHPP) assuming hazard rate is proportional to the remaining number of faults in the software. This research paper was a pioneering attempt in the field of software reliability growth modeling. Later researchers proposed many SRGM, which describe failure, and fault removal phenomenon by NHPP following the basic assumption of GO model and the research is still continuing.

Starting from the late 1960s and in the approximately past 40 years a vast literature of software reliability models has been developed. Extensive research exists in modeling the failure detection and fault removal phenomenon by an NHPP. These models have been widely studied and applied on real software projects. Many concepts of software reliability modeling have been developed and models are proposed considering the various dynamic aspects of the software testing and debugging. In this chapter we will discuss some of the early models. First of all models based on the execution time are discussed. Execution time models prove superior to the calendar time models in many cases [11, 12]. However since the quantities expressed in terms of calendar time component are more meaningful to most engineers and managers most of the models were developed on the calendar time component. Hence we continue our discussion with the calendar time models. Most of these models assume a perfect debugging environment. This means whenever an attempt is made to remove a detected fault it is removed perfectly and no new faults are generated. The earliest model in this category was due to Goel and Okumoto [4]. Some other models in this category are Yamada et al. [13], Ohba [14], Yamada and Osaki [15], Bittanti et al. [16] and Kapur and Garg [17], etc. SRGM which use calendar time or execution time as the unit of fault-detection/removal period either assume that the consumption rate of testing resources is constant, or do not explicitly consider the testing effort and its effectiveness. The achieved reliability during testing phase is highly related to the amount of development resources (test-efforts) spent on detecting and correcting latent software faults. A testing-effort function describes the distribution or consumption pattern of testing resources (CPU time, manpower, etc.) during the testing period. Hence it is very important to track the reliability growth with respect to the testing-effort expenditure. Putnam [18], Yamada et al. [19–21], Bokhari and Ahmad [22], Kapur et al. [23], Kuo et al. [24], Huang [25] and Huang et al. [26, 27] proposed SRGM describing the relationship among the testing time (calendar time), testing-effort expenditure and the number of software faults detected. In the later sections of this chapter we will discuss these SRGM. Models developed considering fault complexity, fault-detection correction process and distributed environment assuming perfect debugging environment are also discussed in this chapter.

2.2 Execution Time Models

Basic execution time model due to Musa [6] and the Logarithmic Poisson execution time model due to Musa and Okumoto [11] are the two most known models in the execution time models category.

2.2.1 The Basic Execution Time Model

The basic execution time model due to Musa [6] is based on the assumptions:

1. The failure process is described by the NHPP.
2. Whenever a failure is experienced the fault causing the failure is removed immediately.
3. The debugging process is perfect.
4. The failure intensity is a decreasing function of execution time.
5. The decrement in failure intensity is constant throughout the testing process.

Notation

λ	Failure intensity
λ_0	Failure intensity at the start of the execution
m ($m(t)$)	Expected number of failures experienced up to time a given time t
a	The total number of that would be experienced in infinite time
t	Execution time

The failure intensity expressed as a function of number of failures experienced is given as

$$\lambda(m) = \lambda_0\left[1 - \frac{m}{a}\right] \tag{2.2.1}$$

Equation (2.2.1) expressed as a function of time can be rewritten as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \lambda(t) = \lambda_0\left[1 - \frac{m(t)}{a}\right] \tag{2.2.2}$$

Solving (2.2.2) under the initial condition $m(0) = 0$, the mean value function of the failure process is given as

$$m(t) = a\left(1 - \mathrm{e}^{-(\lambda_0/a)t}\right) \tag{2.2.3}$$

and the failure intensity of the model is given by

$$\lambda(t) = \lambda_0 \mathrm{e}^{-(\lambda_0/a)t} \tag{2.2.4}$$

2.2.2 The Logarithmic Poisson Model

The assumptions 1–4 are same as those of the basic execution time model. The only difference is in the rate of decrement of the failure intensity, which is assumed to be exponential in contrast to the basic model, i.e. as the testing progresses the decay in the failure intensity decreases. Initially when the fault corresponding to the first failure is repaired a substantial decline is observed in the failure intensity while on the later failures this decrement decreases exponentially.

The failure intensity expressed as a function of number of failures experienced is given as

$$\lambda(m) = \lambda_0 e^{-\theta m} \tag{2.2.5}$$

Equation (2.2.5) can be expressed as a function of time

$$\frac{d}{dt}m(t) = \lambda(t) = \lambda_0 e^{-\theta m(t)} \tag{2.2.6}$$

Solving (2.2.2) under the initial condition $m(0) = 0$ we obtain the mean value function of the failure process as

$$m(t) = \frac{1}{\theta}\ln(1 + \lambda_0 \theta t) \tag{2.2.7}$$

and the failure intensity of the model is given by

$$\lambda(t) = \frac{\lambda_0}{(1 + \lambda_0 \theta t)} \tag{2.2.8}$$

Figures 2.1 and 2.2 show the graphical plots of the failure intensity function and expected number of failures experienced on the mean failures experienced scale and execution time scales, respectively. It has been demonstrated in the

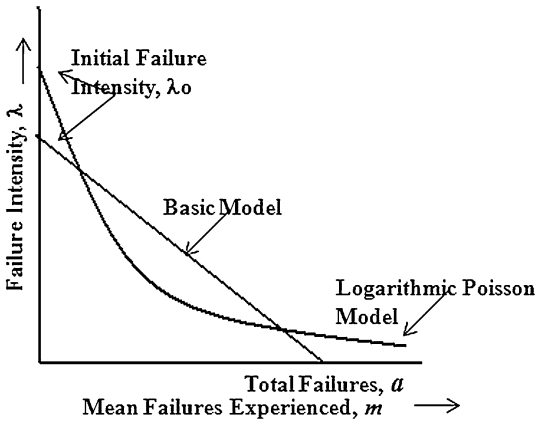


Fig. 2.1 Failure intensity functions

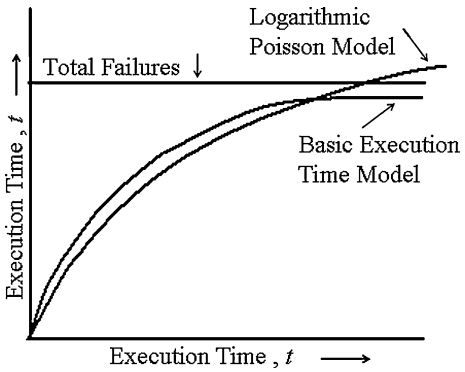


Fig. 2.2 Mean failures experienced w.r.t. execution time

literature [28, 29] that the basic execution time model is applicable to a uniform operational profile whereas Logarithmic Poisson model is applicable under highly non-uniform operational-profile. It may be noted that at infinite execution time, the failure intensity reduces to zero and the number of failures experienced reaches infinity for the logarithmic Possion model even though the number of faults in the software may be finite. This is possible when, either at the time of debugging faults is being introduced, or the debugging process is imperfect or each fault generates more than one failure or when combinations of these Possibilities occurs. It is obvious in this case that the parameters of interest are not the number of errors in software, but the failure intensity and the rate at which failures are occurring.

Both models have been used by Musa to derive some additional quantities of interest to the software engineers and developers. The model is used to determine the additional number of failures that must be experienced (Δm) and the execution time (Δt) is required to reach a failure intensity objective.

If λ_p is the present failure intensity and λ_f is the reqiured failure intensity objective, then for the basic model

$$\lambda_p = \lambda_0\left[1 - \frac{m}{a}\right] \quad \text{and} \quad \lambda_f = \lambda_0\left[1 - \frac{m + \Delta m}{a}\right] \tag{2.2.9}$$

Eq. (2.2.9) yields $\Delta m = \frac{a}{\lambda_0}\left[\lambda_p - \lambda_f\right]$.

And if we write λ_p and λ_f in terms of execution time then

$$\lambda_p = \lambda_0 e^{-(\lambda_0/a)t} \quad \text{and} \quad \lambda_f = \lambda_0 e^{-(\lambda_0/a)(t+\Delta t)} \tag{2.2.10}$$

Eq. (2.2.10) yields $\Delta t = \frac{a}{\lambda_0}\ln\frac{\lambda_p}{\lambda_f}$.

For the Logarithmic Poisson model these quantities can be derived similarly and are given as

$$\Delta m = \frac{1}{\theta}\ln\frac{\lambda_p}{\lambda_f} \quad \text{and} \quad \Delta t = \frac{1}{\theta}\left[\frac{1}{\lambda_f} - \frac{1}{\lambda_p}\right] \tag{2.2.11}$$

Both models have been validated by many researchers [29–31], etc. and have been applied on many real software projects. The models have provided good results (for details see Musa et al. [32]).

2.3 Calendar Time Models

Most of the SRGM proposed in the literature are based on calendar time, as this is the time component, which is more meaningful to the software engineers, developers as well as users. Literature of calendar time model is very vast and as such all models cannot be discussed here in this chapter. In order to facilitate the study and application of these models, they have been categorized into various categories (refer Sect. 1.5.6). We start our discussion in this chapter with the calendar time models, which are earliest proposed and based on the concept of perfect debugging. By *perfect debugging environment* we mean whenever an attempt is made to remove a detected fault it is removed perfectly and no new faults are generated. Various models have been proposed in this category. Besides this we describe here the NHPP models which can be classified as exponential models and *S-shaped models*. Goel and Okumoto exponential model proposed in the year 1979 was recognized as a pioneering attempt. Models developed later, mainly attempted to incorporated more aspects of testing and debugging process and relaxing certain assumptions of the model.

Notation

$m(t)$	Expected number of failures/removals by time t, with $m(0) = 0$
$m_f(t)$	Expected number of failures by time t, with $m_f(0) = 0$
$m_r(t)$	Expected number of removals by time t, with $m_r(0) = 0$
$a(a_i)$	Initial error content in software (or module i/type i) before software testing
$b(t)$	Time-dependent fault-detection/removal rate (FDR/FRR) per remaining faults
$b(b_i)$	Constant fault-detection/removal rate per remaining faults in software (module i/type i) $0 < b < 1$
β, c	Constant parameter in the logistic function

2.3.1 Goel–Okumoto Model

The model [4] is based on the following assumptions:

1. Failure observation phenomenon is modeled by NHPP.
2. Failures are observed during execution caused by remaining faults in the software.

3. Each time a failure is observed, an immediate effort takes place to find the cause of the failure and the isolated faults are removed prior to future test occasions.
4. All faults in the software are mutually independent.
5. The debugging process is perfect and no new fault is introduced during debugging.

Above assumptions can be described mathematically with the following differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = b(a - m(t)) \tag{2.3.1}$$

Solving (2.3.1) with the initial condition $m(0) = 0$ we get

$$m(t) = a\left(1 - \mathrm{e}^{-bt}\right) \tag{2.3.2}$$

The model is known as exponential NHPP model as it describes an exponential failure curve. GO model has been applied to a variety of testing environment in practice. In a number of situations it provides good estimation and prediction of reliability. Hence can be considered as a useful reliability model. The two main aspects of a good model are that the model must be stable during the test period and remain stable until the end of the test phase for any particular test environment and the model must provide a reasonably accurate prediction of the field reliability.

Following, the general assumptions of GO model other exponential SRGM are proposed by Ohba [14] and Yamada and Osaki [15]. Ohba assumed that the software consists of a number of independent modules whereas Yamada and Osaki assumed there are two types of errors in the software. Both these models describe the failure phenomenon for each module\error type by GO model with different parameters and the mean value function is the sum of mean value function for each module\error type.

2.3.2 Hyper-Exponential Model

Assuming that the software is consisting of n independent modules, with different initial number of errors and fault-detection rate for each module, the differential equation describing the model [14] along with general assumptions is

$$\frac{\mathrm{d}m_i(t)}{\mathrm{d}t} = b_i(a_i - m_i(t)) \quad i = 1, 2, \ldots, n \tag{2.3.3}$$

Solution of the above differential equation with the initial condition $m_i(0) = 0$ is similar to the GO model and the mean value function of the failure process for the software can be obtained using

$$m(t) = \sum_{i=1}^{n} m_i(t) = \sum_{i=1}^{n} a_i\left(1 - \mathrm{e}^{-b_i t}\right) \tag{2.3.4}$$

The model is called hyper-exponential model since sum of exponential distributions is hyper-exponential.

2.3.3 Exponential Fault Categorization (Modified Exponential) Model

This model [15] assumes that there are two types of faults in the software—Type I (easy to detect) and Type II (difficult to detect). If q_i is the proportion of type i faults, $i = 1,2$ ($q_1 + q_2 = 1$) the differential equation describing the model is

$$\frac{\mathrm{d}m_i(t)}{\mathrm{d}t} = b_i(ap_i - m_i(t)) \quad i = 1, 2 \tag{2.3.5}$$

The model equation under the initial condition $m_i(0) = 0$ is given by

$$m(t) = \sum_{i=1}^{2} m_i(t) = a\sum_{i=1}^{2} p_i\left(1 - \mathrm{e}^{-b_i t}\right) \tag{2.3.6}$$

Here it is expected that $0 < b_2 < b_1 < 1$.

The early development in the NHPP modeling assumed that the exponential models accounting to the uniform operational profiles could describe the failure process. However the assumption of uniformity seems to be unrealistic in most of the real testing profiles. There can be several reasons of non-uniform testing profile. In order to cater to the need of modeling non-uniform operational profile many researchers later attempted to develop models describing S-shaped failure curve. In most of the real testing situations an S-shaped curve proved ideal to describe the non-uniform operational profile. Several S-shaped SRGM have been proposed in the literature and various authors attributed this S-shapedness to the distinctive explanations.

First of all Yamada et al. [13] refined the GO model describing testing as a two-stage process namely fault-detection and removal. S-shaped SRGM proposed by Ohba [33], Bittanti et al. [16] and Kapur and Garg [17] have similar mathematical forms but are developed under different set of assumptions. An additional important characteristic of the S-shaped models is that most of these models can describe both exponential and S-shaped growth curves depending on the parameter values and therefore are termed as flexible models. This flexible nature of S-shaped models makes them more appropriate for real testing projects. Some S-shaped models are discussed in the following sections.

2.3.4 Delayed S-Shaped Model

This model [13] defines the testing as a two-stage process—failure observation and the corresponding fault removal phenomenon.

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{\mathrm{f}}(t) = b(a - m_{\mathrm{f}}(t)) \tag{2.3.7}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{\mathrm{r}}(t) = b(m_{\mathrm{f}}(t) - m_{\mathrm{r}}(t)) \tag{2.3.8}$$

Solution of the above differential equations under initial condition $m_{\mathrm{f}}(0) = m_{\mathrm{r}}(0) = 0$, is

$$m_{\mathrm{f}}(t) = a\left(1 - \mathrm{e}^{-bt}\right) \tag{2.3.9}$$

$$m_{\mathrm{r}}(t) = a\left(1 - (1 + bt)\mathrm{e}^{-bt}\right) \tag{2.3.10}$$

Model can be derived alternatively in single stage if we assume $(b^2t)/(1 + bt)$ as the fault detection rate (FDR) in (2.3.1) in place of constant FDR b. This model is purely S-shaped.

2.3.5 *Inflection S-Shaped Model*

This model [33] is based on the assumption that the number of detected faults in $(t, t + \Delta t)$ is proportional to the remaining faults at time t, but the proportionality gradually increases in the testing period.

If $u(m)$ is the inflection rate function the model can be expressed by the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \Phi u(m(t))(a - m(t)) \tag{2.3.11}$$

where

Φ is the FDR
m is the number of faults detected $0 \leq m \leq a$

The inflection function is approximated by a linear function with the inflection parameter r, $0 < r \leq 1$

$$u(m) = r + \frac{(1 - r)m}{a} \tag{2.3.12}$$

The inflection function describes two types of faults in the software. Mutually independent faults which gets detected on a test case execution and mutually dependable faults which become detectable on the detection of mutually independent faults.

Now if we assume $\Psi = \frac{(1-r)}{r}$ the model is given by the equation

$$m(t) = a\left[\frac{1 - \mathrm{e}^{-\Phi t}}{1 + \Psi \mathrm{e}^{-\Phi t}}\right] \tag{2.3.13}$$

If $r = 1$ the models describe an exponential failure curve otherwise an S-shaped curve is observed. Hence this is a flexible model.

2.3.6 Failure Rate Dependent Flexible Model

This model has a very important property that it can describe a range of reliability trends observed during testing. Following the assumptions of the GO model the intensity function of the model [16] is described by the following differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = u(m)(a - m(t)) \tag{2.3.14}$$

The FDR $u(m)$ is defined as

$$u(m) = u_{\mathrm{i}} + (u_{\mathrm{f}} - u_{\mathrm{i}})\frac{m(t)}{a} \tag{2.3.15}$$

which is a linear function of the number of faults removed where $u_{\mathrm{i}} > 0$ and $u_{\mathrm{f}} \geq 0$ are the initial and final FDR.

The solution of the model if $u_{\mathrm{f}} \geq 0$ is

$$m(t) = a\left(\frac{u_{\mathrm{i}}(\mathrm{e}^{u_{\mathrm{f}}t} - 1)}{u_{\mathrm{f}} + u_{\mathrm{i}}(\mathrm{e}^{u_{\mathrm{f}}t} - 1)}\right) \tag{2.3.16}$$

On the other hand if $u_{\mathrm{f}} = 0$ we get

$$m(t) = a\left(\frac{u_{\mathrm{i}}t}{1 + u_{\mathrm{i}}t}\right) \tag{2.3.17}$$

Now we have a look at the reliability trends described by the model

If $u_{\mathrm{i}} = u_{\mathrm{f}} = b$ the model reduces to GO model.
If $u_{\mathrm{f}} < u_{\mathrm{i}}$ failure intensity decreases to zero more rapidly than linearly.
If $u_{\mathrm{f}} = 0$ failure intensity is proportional quadratically to the remaining faults.
If $u_{\mathrm{f}} > u_{\mathrm{i}}$ the failure curve has an inflection point, i.e. S-shaped.

In general smaller the ratio $u_{\mathrm{f}}/u_{\mathrm{i}}$, the larger the rate of convergence of failure intensity.

2.3.7 SRGM for Error Removal Phenomenon

This model is proposed by Kapur and Garg [17] (KG model). The model is based on the assumption that the debugging team can also remove some additional errors while removing some errors without these errors causing any failure. The faults

identified on a failure are termed as independent faults while the faults removed additionally are termed as dependent faults.

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{\mathrm{r}}(t) = p(a - m_{\mathrm{r}}(t)) + q\frac{m_{\mathrm{r}}(t)}{a}(a - m_{\mathrm{r}}(t)) \tag{2.3.18}$$

where p and q are the FDR for independent and dependent faults respectively. The mean value function obtained under the initial condition $m_{\mathrm{r}}(0) = 0$ is

$$m_{\mathrm{r}}(t) = a\left[\frac{1 - \mathrm{e}^{-(p+q)t}}{1 + (q/p)\mathrm{e}^{-(p+q)t}}\right] \tag{2.3.19}$$

If $q = 0$ the model reduces to GO model and failure phenomenon is same as removal else the failure phenomenon is given as

$$m_{\mathrm{f}}(t) = \int_0^t p(a - m_{\mathrm{r}}(x))\,\mathrm{d}x = \frac{ap}{q}\ln\left[\frac{p+q}{p + q\mathrm{e}^{-(p+q)t}}\right] \tag{2.3.20}$$

KG model can be derived alternatively if we assume logistic learning fault-detection rate given by $b(t) = \left(b/1 + \beta\mathrm{e}^{-bt}\right)$ with $b = (p + q)$ and $\beta = (q/p)$. This is also a flexible model as it can describe both exponential (If $\beta = 0$) and S-shaped (If $\beta > 0$). The model can also capture a variety of reliability trends depending on the value of parameter β.

Above we have described some S-shaped and flexible SRGM, which are mostly cited and applied, on the software projects. Different models account different reasons of S-shapedness in the models as we can see Yamada model attributes S-shapedness to the time delay between the fault exposure and fault removal. Ohba attribute it to mutual dependency between software faults, according to Bittanti's model it is due to increased fault-detection rate in the later testing phase. Kapur's model advocates S-shapedness due to both fault dependency and debugging time lag.

2.4 SRGM Defining Complexity of Faults

Most of the models discussed in the previous sections were proposed under the assumption that similar testing efforts and testing strategy is required for removing each of the faults. However this assumption may not be true in practice, different faults may require different amount of testing efforts and testing strategy for their removal from the system. In the literature to incorporate this phenomenon faults are categorized as of different types and are analyzed separately. The first attempt in this category was due to Yamada and Osaki [15] who modified GO exponential SRGM assuming that there are two types of faults in the software (see Sect. 2.3.3). This model is based on the observations that during the early stages of the testing phase the testing team removes a large number of simple faults (faults which are

easy to remove) while the hard faults are removed in the later phases of the testing. Hence the model is formulated by superposition of two NHPP models one for each type of fault. The model incorporates the complexity of faults assuming different FRR for each type of faults. FRR for simple faults is expected to be higher than hard fault. However the model assumes a constant FRR for each type of fault. Kareer et al. [34] modified this model assuming time-dependent FRR for both types of faults. Removal phenomenon of simple faults is described by exponential SRGM while an S-shaped SRGM is used for hard faults.

Here we would like the reader to note that complexity of faults is not to be matched with the fault severity. As far as severity of faults is concerned it can be defined in terms of the impact of fault on the system performance. This is of main concern basically in the operational phase when on the observation of a failure in the user environment, the decision to remove the corresponding fault is based on its impact on the system performance. If the impact is very critical it is removed or otherwise its removal may be postponed for a future release of the software. In the testing phase the issue of fault complexity is considered by the researcher in the sense of the time, efforts and testing strategy required to remove the fault.

Kapur et al. [35] introduced generalized Erlang SRGM classifying the faults in the software system as simple, hard and complex faults. It is assumed that the time delay between the failure observation and its subsequent removal represents the complexity of faults. More severe the fault, more the time delay. The model has been extended to *n*-types of faults.

2.4.1 Generalized SRGM (Erlang Model)

For simple faults on the observation of a failure the corresponding fault can be isolated and removed immediately. Hence the mean value function of the failure phenomenon is governed by the following differential equation.

$$\mathrm{d}m_1(t)/\mathrm{d}t = b_1(a_1 - m_1(t)) \tag{2.4.1}$$

$$m_1(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) \tag{2.4.2}$$

Failure detection and removal is modeled as a two stage process for hard faults

$$\mathrm{d}m_{2\mathrm{f}}(t)/\mathrm{d}t = b_2(a_2 - m_{2\mathrm{f}}(t)) \tag{2.4.3}$$

$$\mathrm{d}m_{2\mathrm{r}}(t)/\mathrm{d}t = b_2(m_{2\mathrm{f}}(t) - m_{2\mathrm{r}}(t)) \tag{2.4.4}$$

$$m_2(t) = m_{2\mathrm{r}}(t) = a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) \tag{2.4.5}$$

Fault removal phenomenon for complex faults is modeled as a three-stage process to represent the severity of complex faults

$$\mathrm{d}m_{3\mathrm{f}}(t)/\mathrm{d}t = b_3(a_3 - m_{3\mathrm{f}}(t)) \tag{2.4.6}$$

$$\mathrm{d}m_{3i}(t)/\mathrm{d}t = b_3(m_{3\mathrm{f}}(t) - m_{3i}(t)) \tag{2.4.7}$$

$$\mathrm{d}m_{3\mathrm{r}}(t)/\mathrm{d}t = b_3(m_{3i}(t) - m_{3\mathrm{r}}(t)) \tag{2.4.8}$$

$$m_3(t) = m_{3\mathrm{r}}(t) = a_3\left(1 - \left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)\mathrm{e}^{-b_3 t}\right) \tag{2.4.9}$$

Mean value function for hard and complex faults is expressed by delayed S-shaped and 3-stage Erlang growth curves. The fault removal rates for simple, hard and complex faults is

$$b_1, d_2(t) = \left(b_2^2 t/1 + b_2 t\right) \quad \text{and} \quad d_3(t) = \left(b_3^3 t^2/\left(2\left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)\right)\right)$$

$d_2(t)$ and $d_3(t)$ increase monotonically with t and tend to b_2 and b_3 as $t \to \infty$. Failure curves for hard and complex faults behave similar to the simple faults in the steady state and hence b_2 and b_3 can be assumed equal to b_1 in steady state. It is also observed that $b_1 > d_2(t) > d_3(t)$ in steady state. The mean value function of the SRGM is

$$m(t) = m_1(t) + m_2(t) + m_3(t), \quad a_1 + a_2 + a_3 = a \tag{2.4.10}$$

The mean value function of the SRGM is generalized to include n different types of faults depending upon their severity

$$m(t) = \sum_{i=1}^{n} a_i \left(1 - \mathrm{e}^{-b_i t} \sum_{j=0}^{i-1} \left\{ \left(b_j t\right)^j / j! \right\}\right), \quad \sum_{i=1}^{n} a_i = a \tag{2.4.11}$$

The mean value function of the SRGM describes the joint effect of the type of faults present in the system on the reliability growth. Such an approach can capture the variability in the reliability growth curve due to the errors of different severity depending on the testing environment which enables the management of the software testing to plan and control their testing strategy to tackle each type of fault lying in the system. Another model in this category is due to Kapur et al. [36, 37], which describes the implicit categorization of faults based on their time of detection. However an SRGM should explicitly define the errors of different severity as it is expected that any type of fault can be detected at any point of testing time. Therefore it is desired to study the testing and debugging process of each type of faults separately as in Erlang model. Shatnawi and Kapur [38] also integrated the effect of learning phenomenon of the testing and debugging teams in the Erlang model.

2.4.2 Incorporating Fault Complexity Considering Learning Phenomenon

Mean value function of simple faults is assumed to be same as in the case of generalized Erlang model (Eq. 2.4.2) [38]. Failure detection and removal is again modeled as a two-stage process for hard faults, where

$$\mathrm{d}m_{2\mathrm{f}}(t)/\mathrm{d}t = b_2(a_2 - m_{2\mathrm{f}}(t)) \tag{2.4.12}$$

$$\mathrm{d}m_{2\mathrm{r}}(t)/\mathrm{d}t = b_2(t)(m_{2\mathrm{f}}(t) - m_{2\mathrm{r}}(t)) = \frac{b_2}{1+\beta_2 \mathrm{e}^{-b_2 t}}(m_{2\mathrm{f}}(t) - m_{2\mathrm{r}}(t)) \tag{2.4.13}$$

Here fault removal rate per remaining fault $b_2(t)$ is assumed to be logistic function to describe the learning of the testing team. Solving the above equation the mean value function for hard faults is given as

$$m_2(t) = m_{2\mathrm{r}}(t) = \frac{a_2\left(1-(1+b_2 t)\mathrm{e}^{-b_2 t}\right)}{1+\beta_2 \mathrm{e}^{-b_2 t}} \tag{2.4.14}$$

Fault removal phenomenon for complex faults is modeled as a three-stage process to represent the severity of complex faults assuming fault removal rate per remaining fault $b_3(t)$ to be logistic function to describe the learning of the testing team

$$\mathrm{d}m_{3\mathrm{f}}(t)/\mathrm{d}t = b_3(a_3 - m_{3\mathrm{f}}(t)) \tag{2.4.15}$$

$$\mathrm{d}m_{3\mathrm{i}}(t)/\mathrm{d}t = b_3(m_{3\mathrm{f}}(t) - m_{3\mathrm{i}}(t)) \tag{2.4.16}$$

$$\mathrm{d}m_{3\mathrm{r}}(t)/\mathrm{d}t = b_3(m_{3\mathrm{i}}(t) - m_{3\mathrm{r}}(t)) = \frac{b_3}{1+\beta_3 e^{-b3t}}(m_3(t) - m_{3\mathrm{r}}(t)) \tag{2.4.17}$$

$$m_3(t) = m_{3\mathrm{r}}(t) = a_3\frac{\left(1-\left(1+b_3 t+\left(b_3^2 t^2/2\right)\right)\mathrm{e}^{-b_3 t}\right)}{1+\beta_3 \mathrm{e}^{-b_3 t}} \tag{2.4.18}$$

The mean value functions for hard and complex faults are expressed by delayed S-shaped and 3-stage Erlang growth curves with logistic removal rates. The fault removal rates for simple, hard and complex faults are

$$b_1, d_2(t) = b_2\left(\frac{1}{1+\beta_2 \mathrm{e}^{-b_2 t}} - \frac{1}{1+\beta_2+b_2 t}\right)$$

and

$$d_3(t) = b_3\left(\frac{1}{1+\beta_3 \mathrm{e}^{-b_3 t}} - \frac{1+b_3 t}{1+\beta_3+b_3 t+(b_3 t)^2}\right) \tag{2.4.19}$$

respectively. $d_2(t)$ and $d_3(t)$ increase monotonically with t and tend to b_2 and b_3 as $t \to \infty$. Failure curves for hard and complex faults behave similar to the simple faults in the steady state and hence b_2 and b_3 can be assumed equal to b_1 in steady state. It is also observed that $b_1 > d_2(t) > d_3(t)$ in steady state. The mean value function of the SRGM is

$$m(t) = m_1(t) + m_2(t) + m_3(t), \quad a_1 + a_2 + a_3 = a \tag{2.4.20}$$

The mean value function of the SRGM is generalized to include n different types of faults depending upon their severity

$$m(t) = a_1\left(1 - e^{-b_1 t}\right) + \sum_{i=2}^{n} \frac{a_i}{1 + \beta_i e^{-b_i t}}\left(1 - e^{-b_i t}\sum_{j=0}^{i-1}\left\{\left(b_j t\right)^j / j!\right\}\right), \quad \sum_{i=1}^{n} a_i = a \tag{2.4.21}$$

After the development of Erlang models the concept of fault complexity is integrated by Kapur et al. [39–41] with several other aspects of testing. We will discuss some of these models in the later chapters.

2.5 Managing Reliability in Operational Phase

Most SRGM proposed in the literature are based on the failure pattern observed during the system-testing phase. The software developers are interested in knowing the remaining number of errors in the software at the release time and the field failure rate to track the system performance. The SRGM formulated for the test phase is usually extended to forecast the failure pattern in the operational phase. A major drawback of doing this is that the extension of the model to the operational phase is done under the assumption "system test environment is as close to that of the operational phase". But this is not true for the real life application since how close we claim the test environment is to the operational environment, it cannot be a mirror image. Other realistic issue associated with the operational phase is the non-instantaneous fault removal and possible fix deferral of certain faults depending upon the fault criticality, outsourced code, time of launch of a new release, etc.

Kapur et al. [42] classified the software's into two categories—1. Project type software also called special purpose software, designed for specific applications for known operational environment as specified by the user. However multiple usage of the software is possible within the user environment either at the same or different locations. A simple example is software, which is designed to automate the working of railway reservation system. The developer does not market such software. 2. Product type software also called as general-purpose software, developed for a specific application according to the need of the software users in the market. The need is determined on the basis of market survey, competitive products in the market, R&D, etc. These types of software are sold in the open market. Many distinct users may buy a licensed copy of such software and use it for their own purpose. During the testing phase the type of software under testing does not affect the reliability growth. However, the type of software under consideration influences the reliability growth during the operational phase. Tracking the system performance is not difficult for the project type software since it is in use on a few readily monitored systems under almost constant usage rate. However it is difficult for the product type software due to wide distribution of this type of

software where the usage typically builds to several thousand independent systems.

Kenny [43] argued that the number of failures in operational phase is strongly influenced by the number of users of the software and proposed a model that includes a factor for the usage rate during the operational phase.

2.5.1 Operational Usage Models—Initial Studies

The model [43] is based on the assumption that the average number of failures is a function of the number of encountered defects, which is a function of the number of instructions executed up to the testing time t

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \frac{\mathrm{d}m}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}t} \tag{2.5.1}$$

assuming all remaining defects are equally likely to cause a failure.

The first component $\mathrm{d}m/\mathrm{d}x$ is defined as

$$\mathrm{d}m/\mathrm{d}x = b_1 \tag{2.5.2}$$

The second component $\mathrm{d}x/\mathrm{d}e$ is

$$\mathrm{d}x/\mathrm{d}e = b_2 r \tag{2.5.3}$$

and the third component is

$$\mathrm{d}e/\mathrm{d}t = b_3 t^k \tag{2.5.4}$$

Substituting (2.5.2), (2.5.3) and (2.5.4) in (2.5.1) and solving with the initial conditions $m(0) = 0$ we obtain

$$m(t) = a\left(1 - \mathrm{e}^{-b\left(t^{k+1}/(k+1)\right)}\right), \quad b = b_1 b_2 b_3 \tag{2.5.5}$$

where a is the number of faults at the time of release of the software and r is the remaining number of faults during the field use. Kenny model uses a power function to describe the software usage, which grows as the number of software users increases particularly for product type software. In the marketing literature power function is seldom used for describing the user growth over time. Kapur et al. [44] redefined the Kenny's model using Bass [45] model of innovation diffusion to describe the user growth.

The model [44] is based on statement coverage. The NHPP failure intensity function is formulated as

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \frac{\mathrm{d}m}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}W}\frac{\mathrm{d}W}{\mathrm{d}t} \tag{2.5.6}$$

which can be described as "the number of failures during testing time t is dependent on the number of executed instructions which is a function of software usage during the operational environment up to the time t".

The first component dm/de is defined as

$$\frac{\mathrm{d}m}{\mathrm{d}e} = (b_1 + b_2(m/a))(a - m) \tag{2.5.7}$$

where b_1 is the rate at which remaining faults cause failures, b_2 is the rate of additional fault removal without causing their failure and e is the number of executed instructions.

The second component de/dw is

$$\frac{\mathrm{d}e}{\mathrm{d}W} = b_3 \tag{2.5.8}$$

which defines a constant number of statement execution per additional software usage. This component gives a measure of statement coverage. Substituting (2.5.7) and (2.5.8) in (2.5.6) and solving with the initial conditions $m(0) = 0$ and $W(0) = 0$ we obtain

$$m(t) = \frac{a\left(1 - \mathrm{e}^{-bW(t)}\right)}{1 + \gamma \mathrm{e}^{-bW(t)}}, \quad b = b_3(b_1 + b_2) \quad \text{and} \quad \gamma = \frac{b_2}{b_1} \tag{2.5.9}$$

Software products are similar to the technological products as such the Bass model of innovation diffusion defining the number of adopters of theses products is used to forecast the number of software users in the field. The usage function $W(t)$ is hence modeled as

$$W(t) = N\left(\frac{1 - \exp^{-(p+q)t}}{1 + (q/p)\exp^{-(p+q)t}}\right) \tag{2.5.10}$$

where N is the total number of potential adopters of the software, p is coefficient of innovation and q is the coefficient of imitation.

The equation in (2.5.10) represents the number of adopters of the software and therefore is not suitable to describe the actual usage of the software. With this both models discussed above apply to product type of software. The usage and hence the usage function of project type software is different form that of product type software. Kapur et al. [42] model addresses the reliability growth during the operational phase classifying software as project and product type and linking an appropriate usage function to each type of software considering testing efficiency. This model is discussed Model in Chap. 3.

2.6 Modeling Fault Dependency and Debugging Time Lag

One common assumption of conventional SRGM is that detected faults are immediately removed. In practice, this assumption may not be realistic in software development. Software testing and debugging are very time-consuming

and expensive process. The time to remove a fault depends on the complexity of the detected faults, the skills of the debugging team, the available manpower, or the software development environment, etc. In the general testing environment fault removal may take a longer time after detection. Further more removals are likely in the later phase of the testing as compared to the beginning. Therefore, the time delay in fault-correction process after the detection process can't be ignored. Besides, Ohba [14] conceived that there are two types of faults in the software: mutually independent faults, and mutually dependent faults. KG Model discussed in Sect. 2.3.7 also considers this underlying fault dependency. During testing, faults, which are removed on the detection of a failure, are mutually independent faults or leading fault, while the faults, which get removed during the removal of some leading faults, are called mutually dependent faults. Mutually dependent faults can be removed if and only if the faults leading to them are removed.

Schneidewind [8] first modeled the fault-correction process using a constant delayed fault-detection process. Later, Xie and Zao [46] extended the Schneidewind model to a continuous version by substituting a time-dependent delay function for the constant delay. A key factor of the continuous version of Schneidewind model is the time-dependent delay function, which measures the expected time lag to correct a detected fault. Yamada et al. [13] delayed S-shaped model and fault complexity models discussed earlier are also addressing to the time lag between the fault-correction and detection. Kapur and Younes [47] analyzed the software reliability considering the fault dependency and debugging time lag.

2.6.1 Model for Fault-Correction—The Initial Study

Notation

$m_f(i)$	Mean value function of faults detection process
$m_r(i)$	Mean value function of faults removal/correction process
$m_i(i)$	Mean value function for independent faults
$m_d(i)$	Mean value function for dependent faults
$\lambda(i)$	Failure intensity
i_f	Estimated time to detect cumulative number of faults m_f
i_r	Estimated time to detect cumulative number of faults m_r
a	Initial fault content
b	Fault-detection/removal rate per remaining fault
p_1,p_2	Proportion of independent, dependent faults $p_1 + p_2 = 1$

It assumes that [8] the observations have been made for the number of errors, which have occurred in intervals of unit length, designated by the index, i, from interval 1 through t.

Assumptions

1. The number of errors detected in each time interval is independent of the number of errors detected in any other time interval.
2. The detected error counts have a probability density function of the same form in each time interval but with different means.
3. The mean number of detected errors decreases from interval to interval as a result of the continuing detection and correction of original errors.
4. The rate of error detection in an interval is proportional to the number of errors in the interval.

Following assumption 3 the failure intensity is an exponentially decaying function

$$\lambda(i) = a \exp^{-bi}, \quad a > 0, \quad b > 0 \tag{2.6.1}$$

Equation (2.6.1) implies

$$m_{\mathrm{f}}(i) = \frac{a}{b}\left(1 - \exp^{-bi}\right) \tag{2.6.2}$$

The time estimated to detect a cumulative number of errors m_{f} is derived from (2.6.2) is

$$i_{\mathrm{f}} = \log(a/(a - bm_{\mathrm{f}}))/b \tag{2.6.3}$$

Cumulative mean number of faults corrected has the same form as that of (2.6.2) but will lag the mean number of faults detected by a constant delay Δi. The lag equals the time estimated to correct a number of faults equal to $m_{\mathrm{f}}(i) - m_{\mathrm{r}}(i)$.

Thus for $i \geq \Delta i$

$$m_{\mathrm{r}}(i) = m_{\mathrm{f}}(i - \Delta i) = \frac{a}{b}\left(1 - \exp^{-a(i-\Delta i)}\right) \tag{2.6.4}$$

The lag Δi can be estimated by finding Δi such that the relationship $m_{\mathrm{r}}(i) = m_{\mathrm{f}}(i - \Delta i)$ is satisfied from the empirical data, where i is the time of making a forecast. The time estimated to correct a cumulative number of errors $\mathrm{m_r}$ obtained from (2.6.4) is

$$i_{\mathrm{r}} = \Delta i + \log(a/(a - bm_{\mathrm{r}}))/b \tag{2.6.5}$$

The Schneidewind model assumed that Δi is same for all i, which means that all detected fault, will be corrected after a constant delay of time Δi. Xie and Zao [46] revised this model using a time-dependent delay function Δt that measures the expected delay in correcting a detected fault at any time. However they proposed the revised model in continuous time instead of discrete time as in case of Schneidewind model. In this case the mean value function of the fault-correction process is

$$m_{\mathrm{r}}(t) = m_{\mathrm{f}}(t - \Delta_t) = \frac{a}{b}\left(1 - \exp^{-b(t-\Delta t)}\right) \tag{2.6.6}$$

If Δt is a constant, this model is same as the Schneidewind model. Xie and Zao assumed Δt to be an increasing function on the account of more delays in fault-correction in the later phases of testing given by

$$\Delta t = \frac{\ln(1+ct)}{b}; \quad c \geq 0$$

with this lag function

$$m_{\mathrm{r}}(t) = \frac{a}{b}\left(1 - (1+ct)\exp^{-bt}\right) \tag{2.6.7}$$

2.6.2 Fault Dependency and Debugging Time Lag Model

Fault Dependency and Debugging Time Lag Model due to Kapur and Younes [47] besides NHPP (refer Sect. 2.3.1) assumptions assumes that:

1. The software errors are divided into two categories—(i) leading faults and (ii) dependent faults.
2. The number of errors in the software system is finite and it is the sum of the errors in each category.
3. The number of leading errors removed in time $(t, t + \Delta t)$ is proportional to the number of leading errors remaining.
4. The number of dependent errors removed in time $(t, t + \Delta t)$ is proportional to the number of dependent errors remaining and to the ratio of leading errors removed at time t and the total number of errors.
5. The dependent errors can be removed when the leading error, which they are dependent on, is removed.

Assumption 4 describes the time lag between the error detection and removal. Failure intensity of the independent and dependent faults is given as

$$\mathrm{d}m_{\mathrm{i}}(t)/\mathrm{d}t = b(ap_1 - m_{\mathrm{i}}(t)) \tag{2.6.8}$$

$$\mathrm{d}m_{\mathrm{d}}(t)/\mathrm{d}t = c(ap_2 - m_{\mathrm{d}}(t))(m_{\mathrm{i}}(t-T)/a), \quad a(p_1+p_2) = a \tag{2.6.9}$$

The mean value function obtained under $m_{\mathrm{f}}(0) = m_{\mathrm{r}}(0) = 0$, assuming $T = 0$ are

$$m_{\mathrm{i}}(t) = ap_1\left(1 - \mathrm{e}^{-bt}\right) \quad \text{and} \quad m_{\mathrm{d}} = ap_2\left(1 - \mathrm{e}^{-\left(\left(ap_1\left(1-\mathrm{e}^{-bt}\right) - bcap_1 t\right)/ab\right)}\right) \tag{2.6.10}$$

with

$$m(t) = m_{\mathrm{i}}(t) + m_{\mathrm{d}}(t) \tag{2.6.11}$$

Although this model takes into account the debugging time lag but a closed form solution is obtained for negligible time lag ($T = 0$). Huang and Lin [48] formulated another model based on the assumptions made in Kapur and Younes model, first modeling the time lag phenomenon for the various time dependent lag functions and then integrated the fault dependency in the model.

Fault-detection process [48] is described as in the GO model

$$m_{\mathrm{f}}(t) = a\left(1 - \mathrm{e}^{-bt}\right) \tag{2.6.12}$$

assuming $\phi(t)$ to be the delay factor the fault-correction process is given by

$$m_{\mathrm{r}}(t) = m_{\mathrm{f}}(t - \phi(t)) = a\left(1 - \mathrm{e}^{-b(t-\phi(t))}\right) \tag{2.6.13}$$

Various conventional models (GO model, Yamada delayed S-shaped model, inflection S-shaped model, Yamada Weibull-type testing-effort function model, Logistic growth curve model, etc.) are then derived using the different form of $\phi(t)$. For example if we assume $\phi(t) = 0$ we obtain GO model, if $\phi(t) = (1/b)\ln(1 + bt)$ Yamada delayed S-shaped model is obtained,

For $\phi(t) = (1/b)\ln\left(\frac{(1+\beta)\mathrm{e}^{-bt}}{1+\beta\mathrm{e}^{-bt}}\right)$ inflection S-shaped model is obtained.

Fault dependency and debugging time lag is integrated by substituting the time dependent functional forms of $\phi(t)$ in place of T in the equation describing the failure intensity of dependent faults in the Kapur and Younes model.

Lo and Huang [49] also proposed a general formulation for modeling fault-detection and correction. The proposition given by them is

if $D(t) = \int_0^t \lambda(s)\,\mathrm{d}s$, $C(t) = \int_0^t \mu(s)\,\mathrm{d}s$ and the differential equations for $m_{\mathrm{f}}(t)$ and $m_{\mathrm{r}}(t)$ are

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{\mathrm{f}}(t) = \lambda(t)(a - m_{\mathrm{f}}(t)) \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}t}m_{\mathrm{r}}(t) = \mu(t)(m_{\mathrm{f}}(t) - m_{\mathrm{r}}(t)) \tag{2.6.14}$$

then we have

$$m_{\mathrm{f}}(t) = a\left(1 - \mathrm{e}^{-D(t)}\right) \quad \text{and} \quad m_{\mathrm{r}}(t) = \mathrm{e}^{-C(t)}\int_0^t ac(s)\mathrm{e}^{C(s)}\left(1 - \mathrm{e}^{-D(s)}\right)\mathrm{d}s \tag{2.6.15}$$

with $m_{\mathrm{f}}(0) = 0$ and $m_{\mathrm{r}}(0) = 0$. Here $\lambda(t)$ is the fault-detection rate per remaining fault, and $\mu(t)$ is the fault-correction rate per detected but not corrected fault.

Singh et al. [50] incorporated the learning of the testing team by using a power function for fault-detection and removal rates in Huang and Lin [48] and applied it to the various conventional models (GO model, Yamada delayed S-shaped model, Kapur and Garg model for error removal phenomenon, etc.). Kapur et al. [41] have integrated the concept of fault complexity, fault dependency and debugging time lag.

2.6.3 Modeling Fault Complexity with Debugging Time Lag

Assuming a_1, a_2, a_3; $a_1 + a_2 + a_3 = a$ be the initial fault content of simple, hard and complex faults, simple faults [41] are described by

$$\mathrm{d}m_1(t)/\mathrm{d}t = b_1(a_1 - m_1(t)) \tag{2.6.16}$$

$$m_1(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) \tag{2.6.17}$$

Fault-detection process for hard faults remains same as that of simple faults while the fault-correction process is described by

$$\mathrm{d}m_2(t)/\mathrm{d}t = b_2(a_2 - m_2(t))\left(\frac{m_1(t - \phi(t))}{a_1}\right) \tag{2.6.18}$$

defining

$$\phi(t) = (1/b_1)\ln(1 + b_1 t) \tag{2.6.19}$$

$$m_2(t) = a_2\left(1 - \mathrm{e}^{-\left(\frac{2b_2}{b_1}\left(1-\mathrm{e}^{-b_1 t}\right) - b_2 t\left(1+\mathrm{e}^{-b_1 t}\right)\right)}\right) \tag{2.6.20}$$

For complex fault also the detection process is described by (2.6.17) while the correction process is described by

$$\mathrm{d}m_2(t)/\mathrm{d}t = b_3(a_3 - m_3(t))\left(\frac{m_1(t - \phi(t))}{a_1}\right) \tag{2.6.21}$$

defining

$$\phi(t) = (1/b_1)\ln\left(1 + b_1 t + (b_1 t)^2/2\right) \tag{2.6.22}$$

$$m_3(t) = a_2\left(1 - \mathrm{e}^{-\left(\frac{3b_3}{b_1}\left(1-(1+b_1 t)\mathrm{e}^{-b_1 t}\right) - b_3 t\left(1-\left(1-\frac{b_1 t}{2}\right)\mathrm{e}^{-b_1 t}\right)\right)}\right) \tag{2.6.23}$$

The mean value function of the SRGM is

$$m(t) = m_1(t) + m_2(t) + m_3(t), \quad a_1 + a_2 + a_3 = a \tag{2.6.24}$$

A study of fault-detection process in isolation can be useful to estimate the number of faults detected in the system however an actual estimate of the achieved reliability can be determined by an estimate of the fault removal phenomenon. An ideal model should be able to describe the two processes separately.

The models discussed in the previous sections describe the very basic aspects of the testing process and can be applied to a variety of software test data. However none of these models describes the effect on the pace of testing due to the testing efforts spent. Testing efforts plays a very crucial role on the testing

progress for example at any instance of time during the testing phase testing can be made more rigorous through the additional testing efforts. A model that accommodates the effect of testing effort on the reliability growth often proves to be more useful in the later phases of the testing as it can be used to determine the amount the additional efforts required to reach a specified reliability objective (testing effort control problem, discussed in Chap. 5). In the next section we describe several models for incorporating the effect of testing effort on reliability growth.

2.7 Testing Effort Dependent Software Reliability Modeling

Most of the earlier SRGM were developed based on calendar time or execution time as the unit of fault-detection/removal period and either assume that the consumption rate of testing resources is constant, or do not explicitly consider the testing effort and its effectiveness. A testing effort function describes the distribution or consumption pattern of testing resources (CPU time, manpower, etc.) during the testing period. Putnam [18], Yamada et al. [19–21], Bokhari and Ahmad [22], Kapur et al. [23], Kuo et al. [24], Huang [51] and Huang et al. [26, 27] proposed SRGM describing the relationship among the testing time (calendar time), testing-effort expenditure and the number of software faults detected. Most existing SRGM belong to exponential type models. Kapur et al. [23], Huang [51] and Huang et al. [26, 27] proposed S-shaped testing effort dependent SRGM based on Yamada delayed S-shaped model which also describe the leaning phenomenon of the testing team. Testing effort dependent SRGM to describe the distributed development environments, testing efficiency and fault complexity are proposed by Kapur et al. [41, 42, 52]. Models due to Lin et al. [53] and Gupta et al. [54] were developed considering the changes during the process of testing with respect to test-efforts (change point concept), etc.

Manpower, CPU time and test cases constitute the test-efforts spent on testing. The testing effort function (TEF) discussed in the literature are mainly parametric as they predict development effort using a formula of fixed form parameterized from historical data records.

2.7.1 Rayleigh Test Effort Model

Putnam [18] pioneered the use of Norden Rayleigh Model [55] for the test-effort consumption estimation. It has been empirically determined that the overall life-cycle manpower curve can be well represented by a Rayleigh curve of the type

$$w(t) = 2K\mathrm{e}^{-a^2}at \tag{2.7.1}$$

where

a	$1/2t_d^2$
t_d	Time at which a is maximum
K	Area under the curve in the time interval $[0,\infty]$ and represents the nominal life-cycle effort in man-years
$w(t)$	pdf of the test-effort function

The cumulative distribution of the test-effort function $W(t)$ obtained from (2.7.1) is

$$W(t) = \left(1 - e^{at^2}\right) \tag{2.7.2}$$

The Rayleigh distribution curve slopes upward, levels off into a plateau and then tails off gradually. According to Putnam, the Rayleigh curve depicts the profile of a software development project, with time on the horizontal axis and manpower on the vertical axis. The test-effort curves given above are non-linear and can be linearized dividing (2.7.1) by t and taking the natural logarithm on both sides.

$$\ln\left(\frac{w(t)}{t}\right) = \ln\left(K/t_d^2\right) + \left(-1/\left(2 \cdot t_d^2\right)\right)t^2 \tag{2.7.3}$$

which is a linear equation in t^2 with decreasing slope.

Based on this equation, Putnam stated "if we know the management parameters K and t_d, then we can generate the manpower, instantaneous cost, and cumulative cost of a software project at any time t by using the Rayleigh equation".

2.7.2 Weibull Test Effort Model

Yamada et al. [19–21] claimed that instantaneous testing-effort decreases during the testing life-cycle since it is reasonable to assume that there is a finite limit of resources available to test the software and proposed Weibull-type distribution to describe the TEF having the following three cases.

Exponential curve: The cumulative testing effort consumed in time (0,t] according to exponential curve is

$$W(t) = \alpha(1 - e^{-vt}) \tag{2.7.4}$$

Rayleigh curve: The cumulative testing effort consumed in time (0,t] according to Rayleigh curve is

$$W(t) = \alpha\left(1 - e^{-vt^2/2}\right) \tag{2.7.5}$$

Weibull curve: The cumulative testing effort consumed in time (0,*t*] according to Weibull curve is

$$W(t) = \alpha\left(1 - \mathrm{e}^{-vt^c}\right) \tag{2.7.6}$$

where

α Total amount of test-effort expenditures required by software testing
v, c Scale and shape parameters

When $c = 1$ or 2 the Weibull curve describes the exponential or Rayleigh curve respectively and hence are special cases of Weibull type curve. If $c > 3$ these testing-effort curves have an apparent peak phenomenon (non-smoothly increasing and degrading consumption curve) which is not suitable for many test-effort data sets in practice. Although a *Weibull-type* curve fits well most of the existing data sets but due to the existing peak phenomenon for $c > 3$ is sometimes not advisable to use.

Following the general assumptions of the GO model the NHPP exponential test-effort based SRGM formulated by Yamada et al. [20] is

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) \Big/ w(t) = b(a - m(t)) \tag{2.7.7}$$

The mean value function of the model under the initial conditions $m(0) = W(0) = 0$ is given as

$$m(t) = a\left(1 - \mathrm{e}^{-bW(t)}\right) \tag{2.7.8}$$

On similar lines we can develop Weibull type test-effort based SRGM for the various time dependent SRGM discussed through out the book. For application of model to actual data sets first the testing effort function is fitted to the observed test-effort data and then parameters of the SRGM for failure or removal process is determined with respect to the estimated test-effort data.

Note: The testing-effort functions proposed by Yamada et al. [19] can be derived under the assumption that, "the testing effort consumption rate at any time t during the testing process is proportional to the testing resource available at that time" [56].

Differential equation describing the testing-effort expenditure rate is given by

$$\frac{\mathrm{d}W(t)}{\mathrm{d}t} = v(t)[\alpha - W(t)] \tag{2.7.9}$$

where $v(t)$ is the time-dependent rate at which testing resources are consumed, with respect to remaining available resources. Solving Eq. (2.4.7) under the initial condition $W(t = 0) = 0$, we get

$$W(t) = \alpha\left[1 - \exp\left\{\int_0^t v(x)\,\mathrm{d}x\right\}\right] \tag{2.7.10}$$

Exponential, Rayleigh and Weibull testing-effort functions are obtained when $v(t) = v$, $v(t) = vt$ and $v(t) = vct^{c-1}$, respectively.

2.7.3 Logistic and Generalized Testing Effort Functions

Parr [57] first advocated the use of logistic function to describe the use of resources consumed by the software testing projects. Although Weibull-type testing-effort functions provided good results for estimating the testing resource expenditures for several projects and used for software reliability modeling by various authors, but due to the peak phenomenon for $c > 3$. It is not advisable to use on many occasions. Hence Huang et al. [58] proposed an SRGM using the logistic test-effort function proposed by Pharr. The logistic test-effort function exhibits similar behavior to the Rayleigh curve, except during the early part of the project. The logistic test-effort function over time period $(0,t]$ can be expressed as

$$W(t) = \frac{\alpha}{1 + \beta e^{-ct}} \tag{2.7.11}$$

Using the above test-effort function the current testing-effort consumption is given as

$$w(t) = \frac{d}{dt} W(t) = \frac{c\beta\alpha e^{-ct}}{(1 + \beta e^{-ct})^2} \tag{2.7.12}$$

The parameters α and c have the meaning same as above and β is a constant. The current testing-effort produces a smooth bell-shaped curve, which reaches its maximum value t_{max} when $t_{max} = (1/c) \ln \beta$.

In contrast to the Weibull-type testing-effort function for which $W(0) = 0$, the initial condition for the logistic TEF is, $W(0) \neq 0$. The divergence between the Weibull-type curve and Logistic curve is concentrated in the earlier stages of software development where progress is often least visible and formal accounting procedures for recording the amount of testing-effort applied may not have been instituted. It is possible for us to judge between these models using some statistical test of their relative ability to fit actual failure data, such as adjusting the origin and scales linearly [57]. The testing-effort function (2.4.11) when used in (2.4.6), the mean value function of the SRGM is given by

$$m(t) = a\left(1 - e^{-bW^*(t)}\right); \quad W^*(t) = W(t) - W(0), \quad W(0) = \frac{\alpha}{1 + \beta} \tag{2.7.13}$$

Huang et al. [59] extended the logistic testing-effort function to a generalized form. The generalized logistic testing-effort function has the advantage of relating a work profile more directly to the natural structure of the software development. Therefore, it can be used to pertinently describe the resource consumption during the software development process and get a conspicuous improvement in modeling the distribution of testing-effort expenditures.

The generalized logistic test-effort function is given as

$$W_{\kappa}(t) = \alpha\left(\frac{(\kappa+1)/A}{1+\beta e^{-c\kappa t}}\right)^{1/\kappa} \tag{2.7.14}$$

where

κ is structuring index with large values
A is a constant

If j = 1 and $A = 2$ test-effort function (2.7.14) reduces to (2.7.11) and if we set $A = \kappa + 1$ a generalized and simple testing-effort function is obtained given as

$$W_{\kappa}(t) = \frac{\alpha}{\sqrt[k]{1+\beta e^{-c\kappa t}}} \tag{2.7.15}$$

For (2.7.15) curve reaches its maximum value $t_{\max}$ when

$$t_{\max} = \frac{\ln \beta/\kappa}{c\kappa} \tag{2.7.16}$$

Logistic testing-effort functions have been used by various researchers to formulate SRGM. Kuo et al. [24] used logistic test-effort function to propose several SRGM with constant, non-decreasing and non-increasing fault-detection rates. Huang et al. [27] incorporated the effect of testing-effort consumption function in the Yamada delayed S-shaped model [13].

2.7.4 Log Logistic Testing Effort Functions

Recently, Bokhari and Ahmad [22] presented how to use Exponentiated Weibull, log-logistic and Burr type III curves to describe the testing-effort function.

The Exponentiated Weibull model was originally proposed by Mudholkar and Srivastava [60] for reliability analysis of hardware systems. This model can generate exponential, Rayleigh and Weibull curves as a special case and is given as

$$W(t) = \alpha\left(1 - e^{-\beta t^{c}}\right)^{\theta}; \quad \theta > 0 \tag{2.7.17}$$

The cumulative log-logistic testing-effort function over the time interval [0,t] is given by

$$W(t) = \alpha\left(\frac{(\beta t)^{c}}{1+(\beta t)^{c}}\right) \tag{2.7.18}$$

and the Burr Type XII test-effort function is given as

$$W(t) = \alpha(1 - (1 + (\beta t)^c)^{-m}) \tag{2.7.19}$$

The performance of these test-effort functions is judged on exponential SRGM (2.7.8) from their relative ability to fit actual failure data.

2.7.5 *Modeling the Effect of Fault Complexity with Respect to Testing Efforts Considering Debugging Time Lag*

Assuming a_1, a_2, a_3; $a_1 + a_2 + a_3 = a$ be the initial fault content of simple, hard and complex faults [41], simple faults are described by exponential SRGM assuming no delay in fault-correction process

$$\frac{\mathrm{d}m_1(t)/\mathrm{d}t}{w(t)} = b_1(a_1 - m_1(t)) \tag{2.7.20}$$

$$m_1(t) = a_1\left(1 - \mathrm{e}^{-b_1 W^*(t)}\right); \quad W^*(t) = W(t) - W(0) \tag{2.7.21}$$

Fault-detection process for hard faults remains same as that of simple faults while the fault-correction process is described by

$$\frac{dm_2(t)/dt}{w(t)} = b_2(a_2 - m_2(t))\left(\frac{m_1(W(t) - \Delta W(t))}{a_1}\right) \tag{2.7.22}$$

defining

$$\Delta W(t) = (1/b_1)\ln(1 + b_1 W^*(t)) \tag{2.7.23}$$

$$m_2(t) = a_2\left(1 - \mathrm{e}\left(\frac{2b_2}{b_1}\left(1 - \mathrm{e}^{-b_1 W^*(t)}\right) - b_2 W^*(t)\left(1 + \mathrm{e}^{-b_1 W^*(t)}\right)\right)\right) \tag{2.7.24}$$

For complex fault also the detection process is described by (2.7.21) while the correction process is described by

$$\frac{dm_2(t)/dt}{w(t)} = b_3[a_3 - m_3(t)]\left[\frac{m_1(W(t) - \Delta W(t))}{a_1}\right] \tag{2.7.25}$$

defining

$$\Delta W(t) = (1/b_1)\ln\left(1 + b_1 W^*(t) + (b_1 W^*(t))^2 \Big/ 2\right) \tag{2.7.26}$$

Equation (2.6.22) implies

$$m_3(t) = a_3\left(1 - \mathrm{e}\left(\begin{array}{l}\dfrac{3b_3}{b_1}\left(1 - (1 + b_1 W^*(t))\mathrm{e}^{-b_1 W^*(t)}\right) \\ - b_3 W^*(t)\left(1 - \left(1 - \dfrac{b_1 W^*(t)}{2}\right)\mathrm{e}^{-b_1 W^*(t)}\right)\end{array}\right)\right) \quad (2.7.27)$$

The mean value function of the SRGM is

$$m(t) = m_1(t) + m_2(t) + m_3(t), \quad a_1 + a_2 + a_3 = a$$

In this section we have defined various testing effort functions proposed in literature. Many authors have shown how to incorporate these test-effort functions in software reliability modeling and proposed exponential as well as S-shaped SRGM using them. We can incorporate the effect of testing-effort functions in most of the existing calendar time models. In the later chapters of this book we will discuss some integrated SRGM incorporating the testing-effort functions.

2.8 Software Reliability Growth Modeling Under Distributed Development Environment

Several challenges are faced by the software developers. Size and complexity of the software has increased far beyond the most optimistic forecast. Growth in the abilities to design, test and maintain software is still slower than required. Along with this software users want faster deliveries with discounted cost. Due to these associated reasons software developers tend to develop and maintain the software under distributed development environment (DDE). In a distributed environment the software development is realized by mapping the full set of system requirements across the various sub-systems. These subsystems are integrated to make the complete software. Development of each subsystem (module) is distributed to various teams who develop their piece of software independently. Some of the modules are build on the existing software/modules, modified and extended to engineer the new release while some of them are wholly new components. Role of software reliability growth models is still important in controlling and managing the development of quality software under distributed environment.

The first attempt in software reliability growth modeling under DDE was due to Yamada et al. [61] who assumed that the software is consisting of 'n' used and 'm' newly developed components. The mean value function of the failure phenomenon for the used component was described by an exponential curve (GO model) while it is S-shaped (Yamada delayed S-shaped model) for the newly developed components. The mean value function for the software system is the sum of the mean value functions of its entire components. As such the model can be given as follows

$$m(t) = a\left[\sum_{i=1}^{n} p_i\left(1 - e^{-b_i t}\right) + \sum_{j=1}^{m} p_{n+j}\left\{1 - \left\{1 + b_{n+j}t\right\}e^{-b_{n+j}.t}\right\}\right] \quad (2.8.1)$$

where

p_i is the proportion of faults in the ith component
b_i is the FDR/FRR of the ith component

Kapur et al. [62] formulated a flexible SRGM to describe DDE. They integrated the concept of fault complexity with software reliability modeling for distributed development environment. They assumed faults in the used component to be of simple type and used GO model to describe their failure phenomenon. For newly developed components (m in number) they assumed some of them contain hard faults while faults in the remaining are of complex type. Hence as such the failure and removal phenomenon for these components is described by two- and three-stage process (as in modeling faults of varying complexities, see Sect. 2.4). They also incorporated the learning of the testing team in their model.

2.8.1 *Flexible Software Reliability Growth Models for Distributed Systems*

Along with the general assumptions of NHPP models [62] distinctive assumptions of the model are

1. There is finite number of reused and newly developed software sub-systems.
2. The time delay between the failure observation and its subsequent removal is assumed to represent the complexity of fault. The more severe the fault, more the time delay.
3. Fault removal rate of the reused sub-system is constant.
4. FRR of newly developed components is a logistic learning function as it is expected that the debugging team gets learning as the testing progresses.

Notation

a Total fault content
a_i Initial fault content of ith component
b_i Proportionality constant of FDR/fault isolation rate (FIR) per fault of ith component
$b_j(t)$ Logistic learning FRR of ith component (newly developed)
$m_{ir}(t)$ Mean number of faults removed from ith component by time t
$m_{if}(t)$ Mean number of failures observed in the ith component (newly developed) by time t
$m_{iI}(t)$ Mean number of faults isolated from ith component (newly developed) by time t

η A constant parameter in the logistic learning function
p Number of reused components having simple type of faults
q Number of new components having hard faults
s Number of new components having complex faults

2.8.1.1 Model for Reused Components

It is assumed that the faults in the reused components are simple faults, which can be removed instantly as soon as they are observed. Hence fault removal in reused components is modeled as one-stage processes

$$\frac{\mathrm{d}}{\mathrm{d}t} m_{ir}(t) = b_i(a_i - m_{ir}(t)) \quad i = 0, 1, \ldots, p \tag{2.8.2}$$

Mean value function of fault removal process for the reused components obtained under the boundary condition $m_{ir}(t = 0) = 0$ is

$$m_{ir}(t) = a_i\left(1 - \mathrm{e}^{-b_i t}\right) \tag{2.8.3}$$

2.8.1.2 Model for Newly Developed Components

Software faults in the newly developed software component can be of different complexity. Newly developed components are classified into two categories one containing hard faults and the other containing complex faults.

Components Containing Hard Faults

The fault removal process of these components is modeled as a two-stage process

$$\frac{\mathrm{d}}{\mathrm{d}t} m_{if}(t) = b_i(a_i - m_{if}(t)) \quad i = p + 1, \ldots, p + q \tag{2.8.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} m_{ir}(t) = b_i(t)(m_{if}(t) - m_{ir}(t)) \quad i = p + 1, \ldots, p + q \tag{2.8.5}$$

where

$$b_i(t) = \frac{b_i}{1 + \eta \mathrm{e}^{-b_i t}} \tag{2.8.6}$$

Equation (2.8.6) describes the learning of the debugging process over the testing period. Mean value function of fault removal process for the new components containing hard faults obtained under the boundary condition $m_{if}(t = 0) = m_{ir}(t = 0) = 0$ is

$$m_{ir}(t) = a_i \frac{\left(1 - (1 + b_i t)e^{-b_i t}\right)}{1 + \eta e^{-b_i t}} \quad i = p + 1, \ldots, p + q \tag{2.8.7}$$

Components Containing Complex Faults

These faults require greater time lag between failure observation and removal. Hence a three-stage process is used to describe their removal-observation, isolation and removal.

$$\frac{d}{dt} m_{if}(t) = b_i(a_i - m_{if}(t)) \quad i = p + q + 1, \ldots, p + q + s \tag{2.8.8}$$

$$\frac{d}{dt} m_{il}(t) = b_i(m_{if}(t) - m_{il}(t)) \quad i = p + q + 1, \ldots, p + q + s \tag{2.8.9}$$

$$\frac{d}{dt} m_{ir}(t) = b_i(t)(m_{il}(t) - m_{ir}(t)) \quad i = p + q + 1, \ldots, p + q + s \tag{2.8.10}$$

where $b_i(t)$ is as given by (2.8.6).

The fault removal phenomenon is hence given by the following equation under the boundary condition, $m_{if}(t = 0) = m_{il}(t = 0) = m_{ir}(t = 0) = 0$

$$m_{ir}(t) = a_i \frac{\left(1 - \left(1 + b_i t + \left(b_i^2 t^2/2\right)\right)e^{-b_i t}\right)}{1 + \eta e^{-b_i t}} \tag{2.8.11}$$

2.8.1.3 Modeling Total Fault Removal Phenomenon

Total fault removal phenomenon for the software is superposition of SRGM for 'p' reused and '$q + s$' newly developed components. From Eqs. (2.8.3), (2.8.7) and (2.8.3) the SRGM for software developed under DDE is given as

$$\begin{aligned} m(t) = &\sum_{i=1}^{p} a_i\left(1 - e^{-b_i t}\right) + \sum_{i=p+1}^{p+q} a_i \frac{\left[1 - \{1 + b_i t\}e^{-b_i t}\right]}{1 + \eta e^{-b_i t}} \\ &+ \sum_{i=p+q+1}^{p+q+s} a_i \frac{\left[1 - \left\{1 + b_i t + \frac{b_i^2 t^2}{2}\right\}e^{-b_i t}\right]}{1 + \eta e^{-b_i t}} \end{aligned} \tag{2.8.12}$$

2.8.2 Generalized SRGM for Distributed Systems with Respect to Testing Efforts

Along with the general assumptions of NHPP (refer Sect. 2.3.1) and 1–2 of Sect. 2.8.1 specific assumption of the model [52] is

1. Fault removal rate of the reused sub-system is power function of testing-efforts.
2. Fault removal rate of newly developed components power logistic learning function of testing time as it is expected the debugging team gets learning as the testing progresses.

Additional Notation

a	Total fault content
a_i	Initial fault content of ith component
b_i	Proportionality constant of FDR/fault isolation rate (FIR) per fault of ith component
$b_j(t)$	Power logistic learning FRR of ith component (newly developed)
$m_{ir}(t)$	Mean number of faults removed from ith component by time t
$m_{if}(t)$	Mean number of failures observed in the ith component (newly developed) by time t
$m_{iI}(t)$	Mean number of faults isolated from ith component (newly developed) by time t
η	A constant parameter in the logistic learning function
p	Number of reused components having simple type of faults
q	Number of new components having hard faults
s	Number of new components having complex faults
d	Constant power
$W(t)$, W	Cumulative testing-effort by time t

2.8.2.1 Model for Reused Components

It is assumed that the faults in the reused components are simple faults, which can be removed instantly as soon as they are observed. Hence fault removal of reused components is modeled as one-stage processes

$$\frac{\mathrm{d}m_{ir}(t)/\mathrm{d}t}{w(t)} = b_i W^d(a_i - m_{ir}(t)) \quad i = 0, 1, \ldots, p \tag{2.8.13}$$

Mean value function of fault removal process for the reused components obtained under the boundary conditions $m_{ir}(t = 0) = 0$ and $W(0) = 0$ is

$$m_{ir}(t) = a_i\left(1 - e\left(-\left(b_i W^{d+1}\right)/(d+1)\right)\right) \tag{2.8.14}$$

For different values of d different types of growth curves are captured.

2.8.2.2 Model for Newly Developed Components

Software faults in the newly developed software component can be of different complexity. Newly developed components are classified into two categories: one containing hard faults and the other containing complex faults.

Components Containing Hard Faults

The fault removal process of these components is modeled as a two-stage process

$$\frac{\mathrm{d}m_{if}(t)/\mathrm{d}t}{w(t)} = b_i W^d (a_i - m_{if}(t)) \quad i = p+1, \ldots, p+q \tag{2.8.15}$$

$$\frac{\mathrm{d}m_{ir}(t)/\mathrm{d}t}{w(t)} = b_i(W)(m_{if}(t) - m_{ir}(t)) \quad i = p+1, \ldots, p+q \tag{2.8.16}$$

where

$$b_i(W) = \frac{b_i W^d}{1 + \eta e(-(b_i W^{d+1})/(d+1))} \tag{2.8.17}$$

Equation (2.8.17) describes the learning of the debugging process over the testing period. Mean value function of fault removal process for the new components containing hard faults obtained under the boundary condition $m_{if}(t=0) = m_{ir}(t=0) = W(0) = 0$ is

$$m_{ir}(t) = a_i \frac{\left(1 - \left(1 + \left(b_i W^{d+1}\right)/(d+1)\right) e\left(-\left(b_i W^{d+1}\right)/(d+1)\right)\right)}{1 + \eta e((b_i W^{d+1})/(d+1))} \quad i = p+1, \ldots, p+q \tag{2.8.18}$$

Components Containing Complex Faults

These faults require greater time lag between failure observation and removal. Hence a three-stage process is used to describe their removal-observation, isolation and removal.

$$\frac{\mathrm{d}m_{if}(t)/\mathrm{d}t}{w(t)} = b_i W^d (a_i - m_{if}(t)) \quad i = p+q+1, \ldots, p+q+s \tag{2.8.19}$$

$$\frac{\mathrm{d}m_{il}(t)/\mathrm{d}t}{w(t)} = b_i W^d(m_{if}(t) - m_{il}(t)) \quad i = p+q+1,\ldots,p+q+s \tag{2.8.20}$$

$$\frac{\mathrm{d}m_{ir}(t)/\mathrm{d}t}{w(t)} = b_i(W)(m_{il}(t) - m_{ir}(t)) \quad i = p+q+1,\ldots,p+q+s \tag{2.8.21}$$

where $b_i(t)$ is as given by (2.8.6).

The fault removal phenomenon is hence given by the following equation under the boundary condition, $m_{if}(t=0) = m_{il}(t=0) = m_{ir}(t=0) = W(0) = 0$

$$m_{ir}(t) = a_i \frac{\left(1 - \begin{pmatrix} 1 + \left(b_i W^{d+1}\right)/(d+1) \\ +\left(\left(b_i W^{d+1}\right)/(d+1)\right)^2/2 \end{pmatrix} e\left(-\left(b_i W^{d+1}\right)/(d+1)\right)\right)}{1 + \eta e\left(-\left(b_i W^{d+1}\right)/(d+1)\right)}$$
$$i = p+q+1,\ldots,p+q+s \tag{2.8.22}$$

2.8.2.3 Modeling Total Fault Removal Phenomenon

Total fault removal phenomenon for the software is superposition of SRGM for 'p' reused and '$q+s$' newly developed components and is given by the sum of the mean value functions in Form Eqs. (2.8.14), (2.8.18) and (2.8.22) the SRGM for software developed under DDE is given as

$$\begin{aligned} m(t) = &\sum_{i=1}^{p} a_i\left(1 - \mathrm{e}^{-b_i t}\right) + \sum_{i=p+1}^{p+q} a_i \frac{\left[1 - \{1 + b_i t\}\mathrm{e}^{-b_i t}\right]}{1 + \eta \mathrm{e}^{-b_i t}} \\ &+ \sum_{i=p+q+1}^{p+q+s} a_i \frac{\left[1 - \left\{1 + b_i t + \frac{b_i^2 t^2}{2}\right\}\mathrm{e}^{-b_i t}\right]}{1 + \eta \mathrm{e}^{-b_i t}} \end{aligned} \tag{2.8.23}$$

Other SRGM for distributed development environment analyzing testing coverage and fault complexity are formulated by Kapur et al. [63] and Yadav et al. [64] respectively. We will discuss them in the later chapters.

2.9 Data Analysis and Parameter Estimation

A number of SRGM have been discussed in this chapter. The task of model validation follows the model development process. Once a model has been validated it can be used for practical application. In the model validation the unknown parameters of the developed model are estimated on some past or collected failure data sets and using these estimated parameters estimates are obtained for those data. This process establishes the validity of the model. Successful application of an SRGM depends on a number of factors: majority of them include the testing profile under

consideration, major factors affecting the testing process, characteristic of the selected model, methods used to collect the data and nature, precision and consistency of the collected data. For any practical application some representative models have been selected (see Sect. 1.5.4). Application of these selected SRGM on the observed data involves, first estimating the unknown parameter of the models using the collected data. Our knowledge of statistics helps in this regard. Maximum Likelihood Estimate (MLE) and Non-linear Least Square (NLLS) [65, 66] are the two most widely used estimation techniques. Second, predictive power of the models is judged by estimating the model parameters on the truncated data and predicting it over the remaining. Third step involves determining which model(s) accurately fits data. Various criteria are used in the literature to measure the goodness of fit and predictive power of the models such as Mean Square, Variation, Bias, Mean Square Prediction Error, R^2, AIC, Relative Prediction Error Chi-squared test, etc. The model(s) which describe the testing process accurately are chosen to estimate and predict the failure and removal phenomenon and plotted against observed values on time scale to obtain the goodness of fit curves. This information is useful to quantitatively measure the various aspects of the testing process and environment and further decision making. Detailed discussion on parameter estimation and model validation has been carried in Sects. 1.6 and 1.7.

Most of the NHPP models existing in the literature are non-linear and solutions of their first-order equations are difficult to find using NLLS and MLE and require numerical algorithms to solve. In this book we have used the software package SPSS for model validation and parameter estimation. We have selected some models discussed in the chapter and now we show the process of model validation and parameter estimation on these models.

Failure data set

The interval domain data are taken from Misra [67] in which the number of faults detected per week (38 weeks) is specified and a total of 231 faults were detected. Three types of faults—critical (1.73%), major (34.2%) and minor (64.07%) are present in the software. Mean square of error (MSE) and R^2 are taken as the goodness of fit criteria.

2.9.1 Application of Time Dependent Models

The following models have been selected for illustrating the data analysis and parameter estimation.

Model 1 (M1) The basic execution time model [6]

$$m(t) = \alpha\left(1 - \mathrm{e}^{-(\lambda_0/a)t}\right)$$

Model 2 (M2) Goel–Okumoto model [4]

$$m(t) = a\left(1 - \mathrm{e}^{-bt}\right)$$

Model 3 (M3) Yamada–Osaki model [15]

$$m(t) = a\sum_{1}^{2} p_i\left(1 - \mathrm{e}^{-b_i t}\right)$$

Model 4 (M4) Yamada delayed S-shaped model [13]

$$m_{\mathrm{r}}(t) = a\left(1 - (1 + bt)\mathrm{e}^{-bt}\right)$$

Model 5 (M5) Bittanti's flexible model [16]

$$m(t) = a\left(\frac{u_{\mathrm{i}}(\mathrm{e}^{u_{\mathrm{f}}t} - 1)}{u_{\mathrm{f}} + u_{\mathrm{i}}(\mathrm{e}^{u_{\mathrm{f}}t} - 1)}\right)$$

Model 6 (M6) SRGM for error removal phenomenon [17]

$$m_{\mathrm{r}}(t) = a\left[\frac{1 - \mathrm{e}^{-(p+q)t}}{1 + (q/p)\mathrm{e}^{-(p+q)t}}\right]$$

Model 7 (M7) Generalized SRGM (Erlang) [35]

$$\begin{aligned} m(t) =& a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) + a_3\left(1 - \left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)\mathrm{e}^{-b_3 t}\right); \\ & a_1 + a_2 + a_3 = a(p_1 + p_2 + p_3), \quad p_1 + p_2 + p_3 = 1 \end{aligned}$$

Model 8 (M8) Incorporating fault complexity considering learning phenomenon (flexible Erlang) [38]

$$\begin{aligned} m(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + \frac{a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right)}{1 + \beta_2 \mathrm{e}^{-b_2 t}} \\ + a_3\frac{\left(1 - \left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)\mathrm{e}^{-b_3 t}\right)}{1 + \beta_3 \mathrm{e}^{-b_3 t}} \end{aligned}$$

Model 9 (M9) Xie and Zao model [46]

$$m_{\mathrm{r}}(t) = \frac{a}{b}\left(1 - (1 + ct)\mathrm{e}^{-bt}\right)$$

Table 2.1 Estimation result for model 1 to model 10

Model	Estimated parameters						Comparison criteria	
							MSE	R^2
M1	476 (a)	0.0162 (λ_0)	–	–	–	–	21.23	0.9946
M2	476 (a)	0.0162 (b)	–	–	–	–	21.23	0.9946
M3	615 (a)	0.0083 (b_1)	0.0284 (b_2)	0.7824 (p_1)	0.2176 (p_1)	–	22.60	0.9950
M4	230 (a)	0.1010 (b)	–	–	–	–	133.35	0.9660
M5	587 (a)	0.0133 (u_i)	0.0083 (u_f)	–	–	–	21.28	0.9950
M6	427 (a)	0.0180 (p)	0.0024 (q)	–	–	–	22.47	0.9940
M7	537 (a)	0.0226 (b_1)	0.0140 (b_2)	0.0429 (b_3)	–	–	22.04	0.9950
M8	389 (a)	0.033 (b_1)	0.11275 (b_2)	0.3656 (b_3)	317.14 (β_1)	567.53 (β_2)	16.08	0.9960
M9	8 (a)	0.0161 (b)	0.00001 (c)	–	–	–	23.17	0.9940
M10	668 (a)	0.0162 (b)	0.99999 (c)	0.7108 (p_1)	0.2892 (p_2)	–	23.16	0.9940

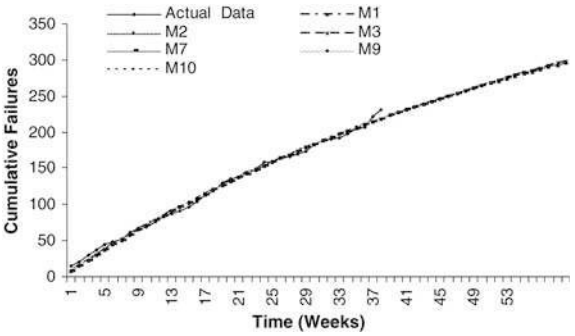


Fig. 2.3 Goodness of fit curve for exponential SRGM (M1–M3, M7, M9, and M10)

Model 10 (M10) Fault dependency and debugging time lag model [47]

$$m_i(t) = ap_1\left(1 - e^{-bt}\right), \quad m_d = ap_2\left(1 - e^{-\left(\left(ap_1\left(1-e^{-bt}\right)-bcap_1t\right)/ab\right)}\right)$$

The unknown parameters of all these models have been estimated using the regression module of SPSS. The values of estimated parameters have been tabulated in Table 2.1. Values in the open brackets are the parameters whose values have been given in the table. Figures 2.3 and 2.4 show the goodness of fit curves for the estimation results shown in Table 2.1 and future predictions of exponential and S-shaped failure curves, respectively.

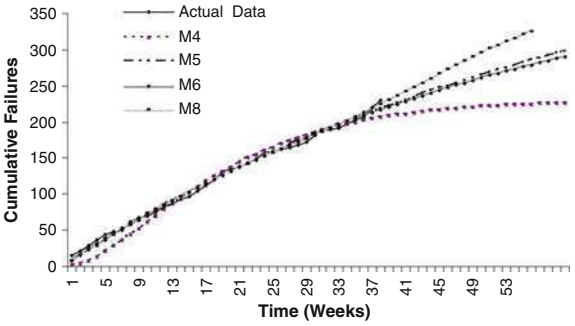


Fig. 2.4 Goodness of fit curve for S-shaped and flexible models (M4–M8)

For the estimation purpose the percentages of critical, major and minor faults have been taken according to the collected data. i.e. $p_1 = 0.6407$, $p_2 = 0.34199$ and $p_3 = 0.01732$. From Table 2.1 we can see that model M8 fits best to this failure data, with MSE = 16.08 and coefficient of determination $R^2 = 0.9960$. As 64% of the faults are of simple type from the data analysis we can say these failure data shows an exponential trend. Pure S-shaped model (M4) (Yamada et al. [13]) is not applicable to this data set. Mean square error is maximum for this model (MSE = 133.35) and R^2 is least (= 0.9660). All other model has given comparable results. The MSE and R^2 values for these models are nearly same. Excluding Yamada et al. [13] model other models taken for the analysis were either exponential or flexible. In case of flexible model Kapur and Garg [17] the result of division of the parameter q and p i.e. $q/p = 0.133$ is very less to add S-shapedness to the estimated failure curve. Same is the case of Bittanti's flexible model. Hence exponential models well describe this data set.

Five exponential models have been considered in this analysis; all of them have been formulated on some different set of assumption. For a practical application one needs to analyze the testing conditions applicable for the testing process under consideration, characteristics and other dimensions of the software project for an accurate selection of the model. For example if we consider the case of this failure data set and we want to measure the reliability of the software with respect to the type of faults present in the software, flexible Erlang model best describes it. The estimation results of this model can now be used to predict the future behavior of the testing process and make decisions such as release time, and resource allocation.

The above data analysis establishes the estimation of the unknown parameters for the various models. Besides estimation results and comparison criteria, selection of a model for practical application is made on the basis of its predictive validity. Estimation results of the data shows an exponential trend and flexible Erlang model gives the best fit. For carrying the predictive validity of any model the observed failure data is truncated in various proportions and using the results of estimation on the truncated data series predictions are made for the remaining data. Now we establish the predictive validity of the flexible Erlang model along with the basic exponential model, Goel and Okumoto [4] model. The results of predictive analysis are tabulated in Tables 2.2 and 2.3.

Table 2.2 Predictive analysis for flexible Erlang model (M8)

Data truncation (%)	Estimated parameters						$m(38)$	RPE	RPE (%)
	a	b_1	b_2	b_3	β_1	β_2			
100	389	0.0340	0.1275	0.3656	317.14	567.53	224	−0.0319	−3.19
95	359	0.0371	0.0841	0.3917	38.73	967.78	219	−0.0516	−5.16
90	347	0.0384	0.0805	0.4315	29.11	967.00	217	−0.0599	−5.99
85	362	0.0360	0.0689	0.2626	19.23	79.70	217	−0.0597	−5.97
80	365	0.0364	0.0820	0.0199	24.27	67.00	224	−0.0305	−3.05
70	372	0.0344	0.0508	0.1845	6.10	35.80	219	−0.0531	−5.31
60	330	0.0390	0.0627	0.3040	6.20	586.31	219	−0.0540	−5.40

Table 2.3 Predictive analysis for Goel and Okumoto model (M2)

Data truncation (%)	Estimated parameters		$m(38)$	RPE	RPE (%)
	a	b_1			
100	475	0.0162	219	−0.0539	−5.39
95	413	0.0192	214	−0.0736	−7.36
90	405	0.0197	213	−0.0766	−7.66
85	409	0.0194	214	−0.0749	−7.49
80	400	0.0200	213	−0.0790	−7.90
70	421	0.0188	215	−0.069	−6.93
60	397	0.0201	212	−0.082	−8.23

Predictive validity of the models is shown on data truncated to 95–60%. Flexible Erlang model shows better predictive validity as compared to Goel and Okumoto [4] model (GO model), although both models underestimate the observed failure data (see column 8 of Table 2.2 and column 4 of Table 2.3). However the level of underestimation is less in case of flexible Erlang model. For both models result shows that even 60% of these observed data is sufficient to predict successfully with relative predictive error (RPE) as low as −5.4% for the flexible Erlang model and −8.23% for the GO model. The overall range of RPE varies from 3 to 6% (approx) for the flexible Erlang model and approximately 5–8% for the GO model, suggesting good predictive validity of both the models. Graphical plot of RPE for both the models is shown in Fig. 2.5.

From the result of predictive validity test we conclude that flexible Erlang model has both good estimating and predictive power for this data set and the information obtained from the analysis can now be used to measure the reliability as well as further decision making.

2.9.2 Application of Test Effort Based Models

The Models chosen above for the numerical analysis and validation were all formulated based on the time (execution/calendar time). We continue data analysis

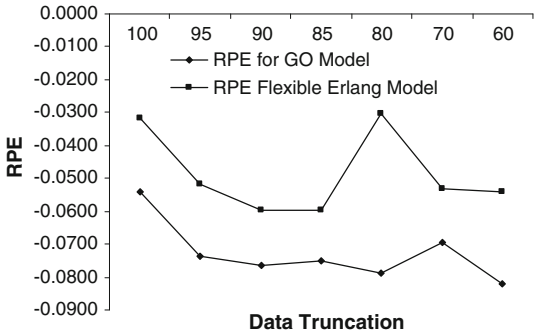


Fig. 2.5 Relative prediction error for the predictive validity test

in this section on the same data set. Now we establish the validity and estimate the unknown parameters of some test-effort based models. The following SRGM have been selected.

Model 11 (M11) Test-effort based Goal Okumoto model [20]

$$m(t) = a\left(1 - \mathrm{e}^{-bW^*(t)}\right); \quad W^*(t) = W(t) - W(0)$$

Model 12 (M12) Test-effort-based fault complexity model considering debugging time lag [41]

$$\begin{aligned} m(t) &= a_1\left(1 - \mathrm{e}^{-b_1 W^*(t)}\right) \\ &+ a_2\left(1 - e\left(\frac{2b_2}{b_1}\left(1 - \mathrm{e}^{-b_1 W^*(t)}\right) - b_2 W^*(t)\left(1 + \mathrm{e}^{-b_1 W^*(t)}\right)\right)\right) \\ &+ a_3\left(1 - e\left(\begin{array}{c} \dfrac{3b_3}{b_1}\left(1 - (1 + b_1 W^*(t))\mathrm{e}^{-b_1 W^*(t)}\right) \\ -b_3 W^*(t)\left(1 - \left(1 - \dfrac{b_1 W^*(t)}{2}\right)\mathrm{e}^{-b_1 W^*(t)}\right) \end{array}\right)\right); \\ W^*(t) &= W(t)) - W(0) \\ a_1 + a_2 + a_3 &= a(p_1 + p_2 + p_3); \quad p_1 + p_2 + p_3 = 1 \end{aligned}$$

First we have estimated the unknown parameters of the exponential, Rayleigh, Weibull and Logistic test-effort functions using the data of cumulative test hours spent in 38 weeks of testing for the data mentioned above. The results of estimation are given in Table 2.4. On the basis of comparison criteria, the results of estimation illustrate that the exponential test-effort function best describes this data set and is hence chosen for further analysis. Using the values of the estimated parameters of the exponential the test-effort function failure curve of the SRGM can be estimated for the past 38 weeks period. Hence using these estimated values parameters of models M11 and M12 are estimated. The results of estimation are

Table 2.4 Estimation results for test-effort functions

Test-effort function	Estimated parameters			Comparison criteria	
	α	v	c, β	MSE	R^2
Exponential	25,887	0.0024	–	3,233.35	0.993
Rayleigh	2,241	0.0040	–	24,957.00	0.946
Weibull	5,063	0.0084	1.1639	4,534.14	0.983
Logistic	2,836	0.0985	10.49	9,751.70	0.98

Table 2.5 Estimation result for model 11 and model 12

Model	Estimated parameters				Comparison criteria	
					MSE	R^2
M2	538 (a)	0.0002 (b)	–	–	17.35	0.995
M8	336 (a)	0.0006 (b_1)	0.0016 (b_2)	0.00025 (b_3)	19.94	0.995

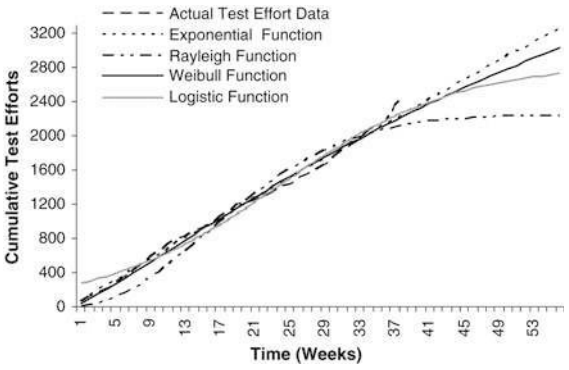


Fig. 2.6 Goodness of fit curve for test-effort functions

shown in Table 2.5. The result illustrates that model M11 fits better to this data set. Further applicability of this model can be made on the basis of the predictive validity results. Establishing the predictive validity of the model M11 is left as an exercise for the readers. Figures 2.6 and 2.7 shows the goodness of fit curves for the test-effort functions and the test-effort- based SRGM.

Exercises

1. Explain the basic difference between an execution time and calendar time based software reliability growth model.
2. The failure curve described by most of the software reliability models is either exponential or S-shaped. State the basic property of testing/operational profile which explains each of these curves.

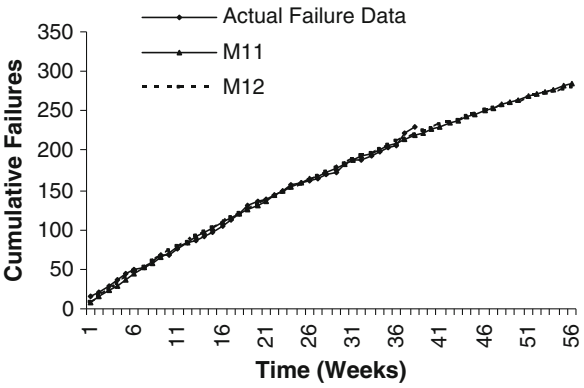


Fig. 2.7 Goodness of fit curve for the test-effort based models

3. Assume that the mean value function of the detection process during testing of a software is described by

$$m_f(t) = a\left(1 - e^{-bt}\right)$$

Obtain the mean value function of the fault repair process if the time lag function for the repair process is given by

(a)

$$\phi(t) = (1/b)\ln(1 + bt)$$

(b)

$$\phi(t) = (1/b)\ln\left(\frac{(1+\beta)e^{-bt}}{1+\beta e^{-bt}}\right)$$

4. What is a testing-effort function? What additional information can be obtained if one chose to describe the testing process of software using an SRGM described with respect to the testing-effort consumed?
5. Give the derivation of the Weibull-type test-effort functions based on the assumption "the testing-effort consumption rate at any time t during the testing process is proportional to the testing resource available at that time".
6. The models described in Sect. 2.3 are all developed based on the calendar time component. Obtain the mean value function of the testing-effort-based SRGM corresponding to the SRGM described in Sects. 2.3.1, 2.3.4 and 2.3.7.
7. Estimate the parameters of the models in Sect. 2.9.1 on a new data set and analyze the result.
8. The data analysis carried in Sect. 2.9.2 suggests that the exponential test-effort function best describes the data used. Fit Exponentiated Weibull test-effort model on the same data and compare the results.

References

1. Haugk G et al (1964) System testing of the no. 1 electronic switching system. Bell Syst Tech J 9:2575–2592
2. Jelinski Z, Moranda P (1972) Software reliability research. In: Freiberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 465–484
3. Moranda P (1975) Predictions of software reliability during debugging. In: Proceedings annual reliability and maintainability symposium, Washington, DC, pp 327–332
4. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R-28(3):206–211
5. Littlewood B, Verrall JL (1973) A Bayesian reliability growth model for computer software. Appl Stat 22:332–346
6. Musa JD (1975) A theory of software reliability and its application. IEEE Trans Softw Eng SE-1:312–327
7. Schneidewind NF (1972) An approach to software in reliability prediction and quality control. In: Fall joint computer conference, AFIPS conference proceedings. AFIPS Press, Montvale, pp 837–847
8. Schneidewind NF (1975) Analysis of error processes in computer software. Sigplan Not 10:337–346
9. Shooman M (1972) Probabilistic models for software reliability prediction. In: Freidberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 485–502
10. Schick GL, Wolverton RW (1973) Assessment of software reliability. Proceedings operations research. Physica-Verlag, Wurzburg Wein, pp 395–422
11. Musa JD, Okumoto K (1984) A logarithmic Poisson execution time model for software reliability measurement. In: Proceedings 7th international conference on software engineering, Orlando, pp 230–237
12. Trachtenberg M (1990) A general theory of software reliability modeling. IEEE Trans Reliab 39(1):92–96
13. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R-32(5):475–484
14. Ohba M (1984) Software reliability analysis models. IBM J Res Dev 28:428–443
15. Yamada S, Osaki S (1985) Software reliability growth modeling: models and applications. IEEE Trans Softw Eng 11:1431–1437
16. Bittanti S, Bolzern P, Pedrotti E, Pozzi N, Scattolini R (1988) A flexible modeling approach for software reliability growth. In: Goos G, Harmanis J (eds) Software reliability modelling and identification. Springer, Berlin, pp 101–140
17. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
18. Putnam LH (1978) A general empirical solution to the macro software sizing and estimating problem. IEEE Trans Softw Eng 4:345–367
19. Yamada S, Ohtera H, Narihisa H (1986) Software reliability growth models with testing-effort. IEEE Trans Reliab R-35:19–23
20. Yamada S, Hishitani J, Osaki S (1991) Test-effort dependent software reliability measurement. Int J Syst Sci 22(1):73–83
21. Yamada S, Hishitani J, Osaki S (1993) Software reliability growth model with Weibull testing effort: a model and application. IEEE Trans Reliab 42:100–105
22. Bokhari MU, Ahmad N (2006) Analysis of software reliability growth models: the case of log-logistic test-effort function. In: Proceedings 7th IASTED international conference on modeling and simulation, Montreal, QC, Canada, pp 540–545

23. Kapur PK, Goswami DN, Gupta A (2004) A software reliability growth model with testing effort dependent learning function for distributed systems. Int J Reliab Qual Safety Eng 11(4):365–377
24. Kuo SY, Huang CY, Lyu MR (2001) Framework for modeling software reliability, using various testing-efforts and fault-detection rates. IEEE Trans Reliab 50(3):310–320
25. Huang CY (2005) Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Softw 76:181–194
26. Huang CY, Lo JH, Kuo SY, Lyu MR (2002) Optimal allocation of testing resources for modular software systems. In: Proceedings 13th IEEE international symposium on software reliability engineering (ISSRE 2002), November 2002, Annapolis, MD, pp 129–138
27. Huang CY, Kuo SY, Lyu MR (2007) An assessment of testing-effort dependent software reliability growth models. IEEE Trans Reliab 56(2):198–211
28. Downs T (1985) An approach to the modeling of software testing with some applications. IEEE Trans Softw Eng 11(4):375–386
29. Trachtenberg M (1985) The linear software reliability model and uniform testing. IEEE Trans Reliab R34(1):8–16
30. Dale CJ (1982) Software reliability evaluation methods. Technical Report ST-26750, British Aerospace Dynamics Group
31. Ramamoorthy CV, Bastani FB (1982) Software reliability status and perspectives. IEEE Trans Reliab 37(1):88–91
32. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York ISBN 0–07-044093-X
33. Ohba M (1984) Inflection S-shaped software reliability growth models. In: Osaki S, Hatoyama Y (eds) Stochastic models in reliability theory. Springer, Berlin, pp 44–162
34. Kareer N, Kapur PK, Grover PS (1990) An S-shaped software reliability growth model with two types of errors. Microelectron Reliab 30(6):1085–1090
35. Kapur PK, Younes S, Agarwala S (1995) Generalized Erlang software reliability growth model. ASOR Bull 35(2):273–278
36. Kapur PK, Bardhan AK, Kumar S (2000) On categorization of errors in a software. Int J Manag Syst 16(1):37–48
37. Kapur PK, Bardhan AK, Shatnawi O (2002) Why software reliability growth modeling should define errors of different severity. J Indian Stat Assoc 40(2):119–142
38. Shatnawi O, Kapur PK (2008) A generalized software fault classification. WSEAS Trans Comput 7(9):1375–1384
39. Kapur PK, Kumar A, Yadav K, Kumar J (2007) Software reliability growth modeling for errors of different severity using change point. Int J Reliab Qual Safety Eng 14(4):311–326
40. Kapur PK, Singh VB, Yang BO (2007) Software reliability growth model for determining fault types. In: Misra RB, Naikan VNA, Chaturvedi SK, Goyal NK (eds) Proceedings 3rd international conference on reliability and safety engineering, INCREASE 2007, Udaipur, pp 334–349
41. Kapur PK, Singh VB, BasirZadeh M (2008) Considering errors of different severity in software reliability growth modeling using fault dependency and debugging time lag functions. In: Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems. MacMillan India Ltd, Bangalore, pp 839–849
42. Kapur PK, Gupta A, Jha PC (2007) Reliability analysis of project and product type software in operational phase incorporating the effect of fault removal efficiency. Int J Reliab Qual Safety Eng 14(3):219–240
43. Kenny GQ (1993) Estimating defects in a commercial software during operational use. IEEE Trans Reliab 42(1):107–115
44. Kapur PK, Bardhan AK, Jha PC (2003) Optimal reliability allocation problem for a modular software system. OPSEARCH J Oper Res Soc India 40(2):133–148

45. Bass FM (1969) A new product growth model for consumer durables. Manag Sci 15:215–227
46. Xie M, Zao M (1992) The Schneidewind software reliability model revisited. In: Proceedings 3rd international symposium on software reliability engineering, pp 184–192
47. Kapur PK, Younes S (1995) Software reliability growth model with error dependency. Microelectron Reliab 35(2):273–278
48. Huang CY, Lin CT (2006) Software reliability analysis by considering fault dependency and debugging time lag. IEEE Trans Reliab 35(3):436–449
49. Lo HJ, Huang CY (2004) Incorporating imperfect debugging into software fault processes. In: TENCON 2004. 2004 IEEE region 10 conference, vol 2, 21–24 November 2004, pp 326–329
50. Singh VB, Yadav K, Kapur R, Yadavalli VSS (2007) Considering fault dependency concept with debugging time lag in software reliability growth modeling using a power function of testing time. Int J Autom Comput 4(4):359–368
51. Huang CY (2005) Cost reliability optimal release policy for software reliability models incorporating improvements in testing efficiency. J Syst Softw 77:139–155
52. Kapur PK, Kumar A, Yadavalli VSS (2006) A general software reliability growth models for a distributed environment. S Afr Stat J 40:151–185
53. Chu-Ti Lin, Chin-Yu Huang, Jun-Ru Chang (2005) Integrating generalized weibull-type testing-effort function and multiple change-points into software reliability growth models. APSEC, pp 431–438
54. Gupta A, Kapur R, Jha PC (2008) Considering testing efficiency in estimating software reliability based on testing variation dependent SRGM. Int J Reliab Qual Safety Eng 15(2):77–81
55. Norden PV (1977) Project life cycle modeling: background and application of the life cycle curves. Presented at the software life cycle management workshop, Airlie, VA sponsored by US Army Computer Systems Command
56. Kapur PK, Gupta A, Shatnawi O, Yadavalli VSS (2006) Testing effort control using flexible software reliability growth model with change point. Int J Performability Eng—Special issue on Dependability of Software/Computing Systems 2:245–262
57. Parr FN (1980) An alternative to the Rayleigh curve for software development effort. IEEE Trans Softw Eng SE-6:291–296
58. Huang CY, Kuo SY, Chen IY (1997) Analysis of a software reliability growth model with logistic testing effort function. In: Proceedings 8th international symposium software reliability engineering (ISSRE'97), pp 378–388
59. Huang CY, Lo JH, Kuo SY, Lyu MR (1999) Software reliability modeling and cost estimation incorporating testing-effort and efficiency. In: Proceedings 10th international symposium software reliability engineering (ISSRE'1999), pp 62–72
60. Mudholkar GS, Srivastava DK (1993) Exponentiated Weibull family analyzing bathtub failure-rate data. IEEE Trans Reliab 42:299–302
61. Yamada S, Tamura Y, Kimura M (2000) A software reliability growth model for a distributed development environment. Electron Commun Japan 83(3):1446–1453
62. Kapur PK, Gupta A, Kumar A, Yamada S (2005) Flexible software reliability growth models for distributed systems. OPSEARCH J Oper Res Soc India 42(4):378–398
63. Kapur PK, Gupta A, Gupta A, Kumar A (2005) Discrete software reliability growth modeling. In: Kapur PK, Verma AK (eds) Quality, reliability and IT (trends and future directions). Narora Publications Pvt Ltd, New Delhi, pp 158–166
64. Yadav K, Goswami DN, Kapur PK (2007) Testing-domain based software reliability growth models for distributed environment. In: Proceedings 3rd international conference on reliability and safety engineering (INCRESE-2007), Udaipur, pp 614–628
65. Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware and software reliability. World Scientific, Singapore
66. Pham H (2006) System software reliability. Reliability engineering series. Springer, London
67. Misra PN (1983) Software reliability analysis. IBM Syst J 22:262–270

Chapter 3
Imperfect Debugging/Testing Efficiency Software Reliability Growth Models

3.1 Introduction

Software systems are developed and designed, for automated functioning of several types of real life functions of the mankind. Even though the creator of software systems is the universe most dominant and intelligent creature, we cannot deny the possibility of software failures during their operational period. These failures are mainly due to the faults manifested in them by their designers. Primarily, testing of software is performed with a goal to detect and remove most of the underlying faults. Even though the software testing and debugging team puts its best efforts, uses distinct methods, engineering techniques, well planed and controlled strategies or the developers make heavy expenditure on testing and debugging; we cannot be assure that the software can be made free of all type of faults at the time of its launch. The faults present in the software at the time of its release can be of three types. First, faults which remained undetected during testing process. Second, faults which were detected and isolated but was not repaired perfectly and third, the faults which got manifested in the software during the removal of some isolated faults. The number of the first type of faults remaining is related to the efficiency of testing which can be reduced by improving the testing methods, testing coverage and resources but still 100% removal cannot be assured due to the constraint of time and testing resources. The second and third types of faults are related to the efficiency of the debugging team, a more skilled debugging team can provide a better debugging service and reduce the number of these types of faults. In software reliability modeling literature, efficiency of debugging team is incorporated as in imperfect debugging software reliability models. Imperfect removal of faults is called as *imperfect fault debugging* and incorporation of new faults during removal of some existing fault is called as *fault generation*.

Quality of test cases, testing environment, testing efficiency and the testing efforts spent are some of the major factors influencing the reliability growth

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_3,
© Springer-Verlag London Limited 2011

during the testing phase. Efficiency and skill of the testing and debugging team greatly influence the testing process since it directly influences the software quality and enables judicious use of testing efforts. Most SRGM formulated in literature assume a perfect debugging environment, i.e. whenever an attempt is made to remove a fault, it is removed perfectly and no new faults are generated; but in most real life situations the debugging process is imperfect due to the reasons described above. However the degree of imperfection may be very low. On the other hand, most NHPP-based SRGM are based on the assumption that failure intensity beside other factors depend on the number of faults remaining in the software. If the imperfect debugging phenomenon is ignored, then an SRGM may provide an optimistic estimate of the remaining fault content resulting in a misleading decision making process later. Thus, as the debugging process plays a very important role in determining the remaining fault content, an SRGM must consider the effect of testing efficiency and debugging process.

During the testing phase, a testing environment close to the operational environment is created and test cases are executed on the software. Any departure from specifications or requirements is termed as a failure. An immediate effort is made to remove the cause of the failure. The fault removal team may not be able to remove the fault perfectly at the detection of a failure and the original fault may remain or be replaced by another fault. To ensure that the cause is perfectly fixed, the software is tested on the same input and if a failure occurs again, the code is checked again. Two possibilities can occur now [1]. The fault, which was thought to be fixed perfectly, has been imperfectly repaired and caused same type of failure again when checked on the same input (imperfect fault debugging-type I). However, it may also happen that some other kind of failure occurs which might be due to the fact that the original fault was perfectly removed but some other fault was generated while removing the cause of the failure (error generation-type II). It may be noted here that generation of a new fault can be known only when the corrected code is retested on the same input. Imperfect fault debugging causes more failures as compared to removals when testing is continued for infinite time period but the fault content remains the same. When faults are generated, the number of failures increases because the fault content has increased. Figure 3.1 summarizes the two types of imperfect debugging phenomenon.

The chapter focuses on software reliability growth models in an imperfect debugging environment. We will describe various SRGM, some of which are purely imperfect fault removal models, some describe error generation only while others describe the integrated effect of two types of imperfect debugging. The concept of imperfect debugging was first introduced by Goel [2]. He introduced the probability of imperfect debugging in Jelinski and Moranda [3] model. Kapur and Garg [4] introduced the imperfect debugging in Goel and Okumoto [5] model. They assumed that the fault removal rate per remaining faults is reduced due to imperfect debugging. Due to error generation phenomena failure count by time infinity becomes more than the initial fault content. These describes the imperfect debugging phenomenon of type I and the software reliability growth curve is exponential. Ohba and Chou [6] introduced the effect of error generation

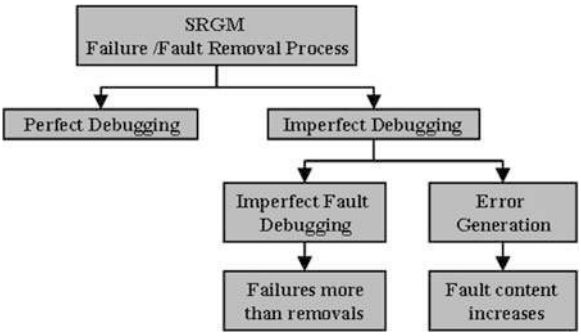


Fig. 3.1 Two types of debugging environments

(imperfect debugging type II) into reliability modeling. Later, based on their model other researchers in the field of reliability modeling further studied the effect of error generation. It may be noted here that generally no distinction have been made between the two types of imperfect debugging during the early stages of research in imperfect debugging phenomenon. All these models have been named as imperfect debugging models even though only one type is incorporated. It has created confusion in providing appropriate insight into the topic as in [7]. Zhang et al. [8] was the first to integrate the two types, modeling it on the number of failures experienced/removal attempts. However in practice a fault is generated while removing some fault and existence of a generated fault is known only after the removal of original fault. Therefore the fault generation rate is expected to be proportional to the rate of fault removals. It may again be noted here that the number of failures is not the same as the number of removals. Kapur et al. [1] comprehensively integrated the two types of imperfect debugging phenomenon in their model and clearly illustrated the facts related to imperfect debugging.

Notation

$m_f(t)$	Mean value function of fault detection process
$m_r(t)$	Mean value function of fault removal/correction process
a ($a_i = ad_i$)	Number of error in the software (or module i/type i) at the time of start of software testing, $\sum d_i = 1$
$a(t)$	Expected initial error content at time t, $a > 0$
p,(p_i)	Probability of perfect debugging of a fault, $0 < p$, ($p_i < 1$
a ($a_i = ad_i$)	Number of error in the software (or module i/type i) at the time of start of software testing, $\sum d_i = 1$
$a(t)$	Expected initial error content at time t, $a > 0$
p,(p_i)	Probability of perfect debugging of a fault, $0 < p$, (p_i) < 1
α,(α_i)	Constant rate of error generation, $0 < \alpha$,(α_i) < 1
$b(t)$	Time dependent rate of fault removal per remaining faults
b, (b_i)	Constant rate of fault detection/removal per remaining faults in software (module i/type i) $0 < b$, (b_i) < 1
β, c	Constant parameter in the logistic function

3.2 Most Primitive Study in Imperfect Debugging Model

Goel [2] first considered effect of testing efficiency in reliability growth during testing. They studied the effect of imperfect debugging on the Markovian model [3]. Jelinski and Moranda model assumes that software faults at the start of testing is a and each fault is independent of others and is equally likely to cause a failure during testing. A detected fault is removed with certainty in a negligible time and no new faults are introduced during the debugging process. The software failure rate, or the hazard function, at any time is assumed to be proportional to the current fault content of the program. In other words, the hazard function during t_i, the time between the $(i-1)$th and ith failures, is

$$Z(t_i) = \phi(a - (i-1)) \tag{3.2.1}$$

where ϕ is a proportionality constant denoting failure rate per fault. The hazard function is constant between failures but decreases in steps of size k (constant) following the removal of each fault. This model assumes that the faults are removed with certainty when detected. To overcome this limitation, Goel [2] proposed an imperfect debugging model as an extension of this model. The number of faults in the system at time t, $X(t)$, is treated as a Markov process whose transition probabilities are governed by the probability of imperfect debugging. The times between the transitions of $X(t)$ are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the $(i-1)$th and ith failures is given by

$$Z(t_i) = \phi(a - p(i-1)) \tag{3.2.2}$$

where p denotes the probability of perfect debugging.

3.3 Exponential Imperfect Debugging SRGM

The models described in this section are based on the general assumptions of NHPP SRGM. Apart from the NHPP assumption, the models here have some additional assumptions on imperfect debugging phenomenon.

3.3.1 Pure Imperfect Fault Debugging Model

In this model [4] it is assumed that on a removal attempt, a fault is removed perfectly with the probability p.

$$\frac{\mathrm{d}m_r(t)}{\mathrm{d}t} = pb(a - m_r(t)) \tag{3.3.1}$$

$$\frac{dm_f(t)}{dt} = b(a - pm_f(t)) \tag{3.3.2}$$

Solving (3.3.1) and (3.3.2) with the initial conditions $m_r(0) = 0$ and $m_f(0) = 0$ we get

$$m_r(t) = a(1 - e^{-bpt}) \tag{3.3.3}$$

$$m_f(t) = (a/p)(1 - e^{-bpt}) \tag{3.3.4}$$

If $p = 1$ the model reduces to GO model with $m_r(t) = m_f(t)$.

3.3.2 Pure Error Generation Model

Ohba and Chou [6] proposed the first SRGM incorporating the effect of error generation based on GO model. They assumed a constant error generation rate α. The following equation describes the failure phenomenon of the model

$$\frac{dm(t)}{dt} = b(t)[a(t) - m(t)] \quad \text{with } b(t) = b,\ a(t) = a + \alpha m(t) \tag{3.3.5}$$

Solving Eq. (3.3.5) under the initial conditions $m(0) = 0$ we get

$$m(t) = (a/1 - \alpha)(1 - e^{-b(1-\alpha)t}) \tag{3.3.6}$$

3.3.3 Using Different Fault Content Functions

Yamada et al. [9] redefined Ohba and Chou [6] SRGM, proposing linear and exponential forms of fault content function.

If we assume $a(t) = ae^{\alpha t}$ in (3.3.5), the mean value function of the SRGM is given as

$$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt}) \tag{3.3.7}$$

otherwise for a linear fault content function $a(t) = a(1 + \alpha t)$, mean value function of the SRGM is given as

$$m(t) = a(1 - e^{-bt})\left(1 - \frac{\alpha}{b}\right) + \alpha at \tag{3.3.8}$$

3.3.4 Imperfect Debugging Model Considering Fault Complexity

Fault categorization in SRGM is an important concept and when it is integrated with the concept of testing efficiency, can provide very accurate estimation and prediction of quality measures. Kapur et al. [10] and Lynch et al. [11] formulated SRGM, integrating the effect of imperfect fault removal and fault generation respectively considering three levels of fault complexity.

3.3.4.1 Pure Imperfect Fault Debugging Model

One stage debugging process of simple faults in the presence of imperfect fault removal is defined by the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{1f}(t) = b_1[a_1 - pm_{1f}(t)] \tag{3.3.9}$$

$$m_{1f}(t) = (a_1/p)(1 - e^{-b_1 pt})$$

$$m_1(t) = m_{1r}(t) = pm_{1f}(t) = a_1(1 - e^{-b_1 pt}) \tag{3.3.10}$$

while the delay in failure detection and removal for hard faults is incorporated as a two stage process defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{2f}(t) = b_2[a_2 - pm_{2f}(t)] \tag{3.3.11}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{2r}(t) = b_2[pm_{2f}(t) - m_{2r}(t)] \tag{3.3.12}$$

$$m_2(t) = m_{2r}(t) = a_2\left(1 - \frac{1}{1-p}e^{-b_2 pt} + \frac{p}{1-p}e^{-b_2 t}\right) \tag{3.3.13}$$

The three stage fault removal process for the complex faults is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{3f}(t) = b_3[a_3 - pm_{3f}(t)] \tag{3.3.14}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{3i}(t) = b_3[pm_{3f}(t) - m_{3i}(t)] \tag{3.3.15}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{3r}(t) = b_3[m_{3i}(t) - m_{3r}(t)] \tag{3.3.16}$$

$$m_3(t) = m_{3r}(t) = a_3\left(1 - \frac{1}{(1-p)^2}(e^{-pb_3 t} - e^{-b_3 t}) - \frac{(1 - p(1 + b_3 t))}{1-p}e^{-b_3 t}\right) \tag{3.3.17}$$

The fault removal rate per fault for simple, hard and complex faults is

$$b_1, \quad d_2(t) = \frac{pb_2(1 - e^{-b_2(1-p)t})}{1 - pe^{-b_2(1-p)t}}$$

$$\text{and} \quad d_3(t) = \frac{pb_3(1 - (1 + b_3(1-p)t)e^{-b_3(1-p)t})}{1 - p((1-p) + (1 + b_3(1-p)t))e^{-b_3(1-p)t}}$$

$d_2(t)$ and $d_3(t)$ increase monotonically with t and tend to pb_2 and pb_3 as $t \to \infty$. Failure curves for hard and complex faults behave similar to the simple faults in the steady state and hence pb_2 and pb_3 can be assumed equal to pb_1 in steady state. It is also observed that $b_1 > d_2(t) > d_3(t)$ in steady state. The mean value function of the SRGM is

$$m(t) = m_1(t) + m_2(t) + m_3(t), \quad a_1 + a_2 + a_3 = a \tag{3.3.18}$$

The mean value function of SRGM describes the joint effect of the type of faults present in the system on the reliability growth in the presence of imperfect repair facilities.

3.3.4.2 Pure Error Generation Model

Lynch et al. incorporated the consideration of three types of faults in the software in Ohba and Chou [6] fault generation SRGM. Fault detection rates are defined as b_i, $i = 1, 2, 3$ for simple, hard and complex faults respectively. Their model can be described by the following differential equations

$$\frac{\mathrm{d}m_i(t)}{\mathrm{d}t} = b_i[a_i(t) - m_i(t)] \tag{3.3.19}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}a_i(t) = \alpha_i \frac{\mathrm{d}}{\mathrm{d}t}m_i(t) \tag{3.3.20}$$

Solving the above equations under the initial conditions $a_i(0) = ad_i$ and $m_i(0) = 0$, mean value function of the SRGM for simple, hard and complex faults is given as

$$m_i(t) = (ad_i/1 - \alpha_i)(1 - e^{b_i(1-\alpha_i)t}) \quad i = 1, 2, 3 \tag{3.3.21}$$

Mean value function of the total failure detection process is given as

$$m(t) = \sum_{i=1}^{3} m_i(t) = \sum_{i=1}^{3} (ad_i/1 - \alpha_i)(1 - e^{b_i(1-\alpha_i)t}) \tag{3.3.22}$$

where d_i is the proportion of the ith type of fault and $\sum_{i=1}^{3} d_i = 1$.

3.3.5 Modeling Error Generation Considering Fault Removal Time Delay

Lo and Huang [12] integrated error generation phenomenon in the SRGM, considering the time lag between failure detection and correction. They claimed that an SRGM which considers both imperfect debugging and removal time delay can often provide much realistic estimates as compared to the models assuming instantaneous fault removal. They modified their generalized detection correction model (see Section 2.5.2, Lo and Huang [12]) incorporating the constant probability of error generation. Assuming each time a failure occurs, the new fault may be introduced in the fault correction process with a probability α, the differential equations for failure detection and correction process are defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_f(t) = \lambda(t)[a(t) - m_f(t)]$$

and

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = \mu(t)[m_f(t) - m_r(t)] \tag{3.3.23}$$

where

$$\frac{\mathrm{d}}{\mathrm{d}t}a(t) = \alpha\frac{\mathrm{d}}{\mathrm{d}t}m(t)$$

The initial conditions $m_f(0) = 0$ and $m_r(0) = 0$ provide the solution

$$a(t) = (a/1-\alpha)(1 - \alpha e^{-(1-\alpha)D(t)}) \tag{3.3.24}$$

$$m_f(t) = (a/1-\alpha)(1 - e^{-(1-\alpha)D(t)})$$

and

$$m_r(t) = a/(1-\alpha)e^{-C(t)}\int_0^t c(s)e^{C(s)}\left(1 - e^{-(1-\alpha)D(s)}\right)ds$$

An application of the approach is shown assuming $\lambda(t) = \lambda$ and $\mu(t) = \mu$, the mean value functions for the failure detection and correction processes are given as

$$m_f(t) = (a/1-\alpha)(1 - e^{-(1-\alpha)\lambda t}) \tag{3.3.25}$$

$$m_r(t) = \frac{a}{(1-\alpha)}\left(1 + \frac{\mu}{\lambda(1-\alpha)-\mu}e^{-(1-\alpha)\lambda t} - \frac{\lambda(1-\alpha)}{\lambda(1-\alpha)-\mu}e^{-\mu t}\right) \tag{3.3.26}$$

3.4 S-Shaped Imperfect Debugging SRGM

Incorporation of imperfect debugging phenomenon in software reliability modeling provided promising improvement in reliability estimation and prediction, together with better understanding of some aspects of testing and debugging, such as current testing efficiency, requirements for improvement, expertise of testing and debugging teams, etc. Earlier studies in imperfect debugging modeling were mainly focused on exponential SRGM, but due to unrealistic assumption of a uniform testing profile of exponential models, need of s-shaped and flexible models was created. Studies in modeling the error generation phenomenon due to Kapur and Younes [13] first satisfied this need. The s-shaped imperfect debugging SRGM gained popularity by the work of Pham et al. [14]. Pham et al. [14] proposed a generalized imperfect debugging fault generation SRGM with s-shaped fault-detection rate in order to incorporate learning of the debugging team. The assumption that failure intensity depends on the remaining fault content, necessitates the incorporation of debuggers learning in the SRGM. Learning is a universal phenomenon and when the experience of the debuggers with the code grows, it is likely that they remove more faults with increasing efficiency in the later stages of the testing phase. On this account an SRGM ignoring the learning phenomenon may provide a pessimistic estimate of the remaining fault content. Pham et al. [14] compared their models with the existing exponential imperfect debugging and some perfect debugging models. Fairly good results were obtained for their models and with improved estimates.

3.4.1 An S-Shaped Imperfect Debugging SRGM

This is pure imperfect fault repair model which assumes the probability of imperfect debugging, $p(t)$ decreasing with testing time i.e. learning occurs with testing progress and probability of imperfect debugging is proportional to the number of errors remaining in the software. Under the other assumptions of NHPP SRGM, the differential equation for the model is formulated as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = b(a - m_r(t)) - cp(t)(a - m_r(t)); \quad b > c > 0 \tag{3.4.1}$$

The model describes the removal intensity as intensity of perfect debugging minus intensity of imperfect debugging. To account, the decreasing probability of imperfect debugging, the $p(t)$ is defined as

$$p(t) = \frac{(a - m_r(t))}{a} \tag{3.4.2}$$

which decreases with time as remaining fault content $(a - m_r(t))$ decreases. The product $cp(t)$ gives the instantaneous imperfect debugging rate. This rate is maximum at the beginning of the testing phase (equal to c) and tends to its

minimum value (equal to zero) when all the original errors are removed. Further, when the debugging is perfect, then $c = 0$ and the model reduces to GO model. Solving Eq. (3.4.1) using (3.4.2) with the boundary condition $m_r(0) = 0$, we obtain

$$m_r(t) = a\left[\frac{(b-c)(1-e^{-bt})}{(b-c)+ce^{-bt}}\right] \tag{3.4.3}$$

and the failure phenomenon is given as

$$m_f(t) = \int_0^t b(a - m_r(x))\,dx = \frac{ab}{c}\ln\left[\frac{b}{(b-c)+ce^{-bt}}\right] \tag{3.4.4}$$

It can be noted here that $m_f(0) = m_r(0) = 0, \quad m_f(\infty) = (a\frac{b}{c})\ln\left(\frac{b}{b-c}\right)$ and $m_r(\infty) = a$. Here $m_f(\infty)$ is greater than a showing the presence of an imperfect debugging efficiency.

3.4.2 General Imperfect Software Debugging Model with S-Shaped FDR

Pham et al. [14] first proposed the general solution for the differential equation (3.3.5) describing Ohba and Chou [6] model. General solution of Eq. (3.3.5) is

$$m_r(t) = \left(e^{-\int_{t0}^{t} b(u)\,du}\right)\left[m_0 + \int_{t0}^{t} a(t)b(t)e^{\int_{t0}^{t} b(u)\,du}\,\mathrm{d}t\right] \tag{3.4.5}$$

where $m_0 = m(t_0)$, t_0 is the time point when testing starts.

The model is analyzed for linear fault content function and non-decreasing s-shaped fault detection rate per remaining fault.

$$a(t) = a(0)(1+\alpha t) \quad \text{and} \quad b(t) = \frac{b(0)(1+\beta)}{1+\beta e^{-b(0)(1+\beta)t}} \tag{3.4.6}$$

$a(0)$, $b(0)$ are defined as initial fault content and initial per fault visibility. Mean value function of the SRGM under the initial condition $m(0) = 0$ is

$$m(t) = \frac{a(0)(1+\beta)}{1+\beta e^{-b(0)(1+\beta)t}}\left(\left(1-\frac{\alpha}{b(0)(1+\beta)}\right)(1-e^{-b(0)(1+\beta)t}) + \alpha t\right) \tag{3.4.7}$$

Later Pham [15] analyzed this model with

$$a(t) = ae^{ct} \quad \text{or} \quad a(t) = k + a(1-e^{-\alpha t})$$

and

$$b(t) = \frac{b}{1+\beta e^{-bt}} \tag{3.4.8}$$

Following are the mean value function for the SRGM obtained with these fault content functions.

$$m(t) = \frac{ab}{b+c}\left(\frac{e^{(b+c)t}-1}{e^{bt}+\beta}\right) \tag{3.4.9}$$

$$m(t) = \frac{1}{1+\beta e^{-bt}}\left((k+a)(1-e^{-bt}) - \frac{a}{b-\alpha}(e^{-\alpha t}-e^{-bt})\right) \tag{3.4.10}$$

Kapur et al. [16] investigated the effect of imperfect debugging for the various forms of fault generation function ($a(t)$) on Yamada et al. [17] delayed S-shaped model. They also incorporated the learning phenomenon in these models.

3.4.3 Delayed Removal Process Modeling Under Imperfect Debugging Environment

Kapur et al. [16] defined delayed s-shaped Yamada et al. [17] model in the imperfect debugging environment using two forms of fault content function.

Assuming that fault generation rate at any time t is proportional to the fault removal rate at that time i.e. $a(t) = a + \alpha m(t)$ in Yamada et al. [17] model, mean value function of the failure and fault removal phenomena are obtained as

$$m_f(t) = \left(\frac{a}{1-\alpha}\right)\left(1 + \frac{1+\sqrt{\alpha}}{2}e^{-b(1-\sqrt{\alpha})t} - \frac{1-\sqrt{\alpha}}{2}e^{-b(1+\sqrt{\alpha})t}\right) \tag{3.4.11}$$

and

$$m_r(t) = \left(\frac{a}{1-\alpha}\right)\left(1 - \frac{1+\sqrt{\alpha}}{2\sqrt{\alpha}}e^{-b(1-\sqrt{\alpha})t} + \frac{1-\sqrt{\alpha}}{2\sqrt{\alpha}}e^{-b(1+\sqrt{\alpha})t}\right) \tag{3.4.12}$$

On the other hand if a linear function of testing time is used to describe the fault content, then mean value functions for failure and fault removal phenomena are given as

$$m_f(t) = a\left(\left(1-\frac{\alpha}{b}\right)(1-e^{-bt}) + \alpha t\right) \tag{3.4.13}$$

and

$$m_r(t) = a\big(1-(1+bt)e^{-bt} - 2(\alpha/b)(1-e^{-bt}) + \alpha t(1+e^{-bt})\big) \tag{3.4.14}$$

Authors have also incorporated the learning phenomenon in this model. The removal phenomenon in the presence of learning is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = \frac{1}{1-\beta e^{-bt}}(m_f(t) - m_r(t)) \tag{3.4.15}$$

with logistic per fault removal rate, the mean value function for the removal phenomenon instead of (3.4.11) is given as

$$m_r(t) = \frac{a}{1-\beta e^{-bt}}(1-(1+bt)e^{-bt} - 2(\alpha/b)(1-e^{-bt}) + \alpha t(1+e^{-bt})) \quad (3.4.16)$$

Authors have defined these models for different fault detection and removal rates.

Kapur et al. [16] also proposed an alternative derivation of delayed s-shaped Yamada et al. [17] (see Sect. 2.2.3) model to obtain it in single stage, assuming

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t} = b(t)[a - m(t)] \quad \text{where } b(t) = \frac{b^2 t}{1+bt} \quad (3.4.17)$$

mean value function of Yamada et al. [17] model can be obtained. The $b(t)$ in (3.4.14) can be considered as a learning fault detection rate.

The model is redefined, considering error generation for different fault content functions i.e.

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t} = b(t)[a(t) - m(t)]$$

If $a(t) = a + \alpha m(t)$ the mean value function of the SRGM is

$$m(t) = \left(\frac{a}{1-\alpha}\right)\left(1-(1+bt)^{1-\alpha}e^{-b(1-\alpha)t}\right) \quad (3.4.18)$$

and if $a(t) = a(1 + \alpha t)$, mean value function of the SRGM is [18]

$$m(t) = a\left(1+\alpha t - \frac{1+bt}{e^{bt}}\right) - \frac{a\alpha(1+bt)}{be^{bt}}\left(\ln(1+bt) + \sum_{i=0}^{\infty}\frac{(1+bt)^{(i+1)}-1}{(i+1)!(i+1)}\right) \quad (3.4.19)$$

All s-shaped models that have logistic fault removal rates are also flexible since when the parameter $\beta = 0$, the SRGM reduces to an exponential type.

3.5 Integrated Imperfect Debugging SRGM

The SRGM discussed in the previous sections incorporates the effect of only one type of imperfect debugging, either imperfect fault removal or fault generation. The fact that both types of imperfect debugging may occurs simultaneously cannot be ignored. On a fault removal occasion, a debugger can both repair the fault incorrectly as well as introduce a new fault. As SRGM considers only imperfect fault removal may give an estimate that the number of faults removed in the total

testing time equals the initial fault content ignoring the generation of faults during debugging. Hence an optimistic estimate of reliability will be obtained. On the other hand if only error generation is considered, more than one failure can correspond to a single fault, resulting in more failures than removals. An estimate of reliability based on failure observation may again be optimistic. It means even if we are incorporating the effect of one type of imperfect debugging ignoring the other type we may get an optimistic estimate of reliability. In order to obtain a more accurate estimate of reliability, one must integrate the effect of the two types of imperfect debugging simultaneously in the software reliability growth modeling. Few attempts have been made in the literature for developing comprehensive models incorporating the two types. We describe these models in this section of the chapter.

3.5.1 Testing Efficiency Model

Zhang et al. [8] integrated the effect of two types of imperfect debugging by simultaneously considering the assumption of Kapur and Garg [4] pure imperfect fault repair model and Ohba and Chou [6] fault generation SRGM. Assuming a constant fault generation rate α, proportional to the failure intensity and constant probability of perfectly debugging p, the model is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = b(t)(a(t) - pm(t)) \tag{3.5.1}$$

where $a(t) = a + \alpha m(t)$ and $b(t) = \left(\frac{c}{(1+\beta e^{-bt})}\right)$.

Mean value function of the failure phenomenon under the initial condition $m(0) = 0$ is

$$m(t) = \frac{a}{p-\alpha}\left(1 - \left(\frac{(1+\beta)e^{-bt}}{1+\beta e^{-bt}}\right)^{(c/b)(p-\alpha)}\right) \tag{3.5.2}$$

In the above model Zhang et al. [8] assumed that the fault generation rate is proportional to the failure intensity in the presence of the possibility of imperfectly removing a fault. However in practice a fault is generated while removing some fault and existence of a generated fault is known, when original fault is removed perfectly and the same test case that has caused the failure is executed to test the corrected code. On the execution of this test case some other kind of failure is observed. Therefore the fault generation rate should be defined proportional to the rate of fault removal. The number of failures is not the same as the number of removals if we assume that the debugging team may not be able to repair a fault perfectly. The same fault may manifest on testing the corrected code for the same input on imperfect removal. Testing efficiency model due to Kapur et al. [1] corrected this ambiguity in modeling the two types of imperfect debugging.

3.5.2 Integrated Exponential and Flexible Testing Efficiency Models

An generalized integrated testing efficiency model considering two types of imperfect debugging is proposed by Kapur et al. [1]. Assuming that fault removal rate per additional fault removed is reduced by the probability of perfect debugging and a constant proportion of removed faults are generated while removal, the general differential equation describing the failure and removal phenomenon is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = pb(t)[a(t) - m_r(t)] \tag{3.5.3}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}a(t) = \alpha(t) \cdot \frac{\mathrm{d}}{\mathrm{d}t}m_r(t) \tag{3.5.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_f(t) = b(t)[a(t) - pm_f(t)] \tag{3.5.5}$$

Equations (3.5.3) and (3.5.5) establishes the relation

$$m_r(t) = pm_f(t) \tag{3.5.6}$$

Under the initial condition $m_r(0) = 0$ and $a(0) = 0$ mean value function of the removal phenomenon can be obtained from the following equation:

$$m_r(t) = e^{-p\int(1-\alpha(u))b(u)\,du}\left[ap\int_0^t b(x)e^{-p\int(1-\alpha(x))b(x)\,dx}\,dx\right] \tag{3.5.7}$$

and the failure phenomenon can be described using (3.4.6) and (3.4.7). For the different forms of fault detection and removal rates per fault and error generation functions, distinct integrated imperfect debugging SRGM can be obtained for the perfect debugging model. Few special cases of the model have been analyzed by the authors.

If a constant fault detection, removal and error generation rate describes the failure and removal processes and error generation i.e. $b(t) = b$ and $\alpha(t) = \alpha$, an imperfect debugging model is obtained for GO model [19], given as

$$m_r(t) = (a/(1-a))(1 - e^{-bp(1-\alpha)t}) \tag{3.5.8}$$

In order to incorporate the efficiency of testing and debugging teams learning forms of $b(t)$ are used i.e.

$$b(t) = \frac{b}{(1+\beta e^{-bt})} \quad \text{and} \quad b(t) = \frac{b^2 t}{1+bt}$$

with $a(t) = a + \alpha m_r(t)$.

Now the mean value function of the removal processes corresponding to the two learning functions are given as

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta)e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \tag{3.5.9}$$

and

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - (1+bt)^{p(1-\alpha)} e^{-bp(1-\alpha)t}\right] \tag{3.5.10}$$

The mean value function of the failure phenomena are derived from the relation $pm_f(t) = m_r(t)$.

It may be noted here that for (3.5.9), $m_r(\infty) = a/(1-\alpha)$ and $m_f(\infty) = a/(p(1-\alpha))$ whereas in Zhang's testing efficiency model $m_f(\infty) = a/(p-\alpha)$, which implies that if testing is carried out for an infinite time more faults are removed as compared to the initial fault content because there are some errors added to the software due to error generation. The total number of generated faults by time infinity as given by Kapur et al. model is $a(t=\infty) - a = (a\alpha)/(1-\alpha)$ whereas for Zhang's model it is $(a(1-p+\alpha))/(p-\alpha)$. It is important to note that imperfect repair of faults results in more number of failures than removals and has no contribution to increasing the fault content. Whereas, the result of Zhang's models yields the total number of generated errors as a function of both p and α.

Kumar et al. [20] integrated the concept of delayed fault removal process in their generalized two type imperfect debugging model. Such a comprehensive model can prove to be very useful in real life applications as it does not make simplistic assumptions on many distinct aspects. The effect of imperfect debugging, learning phenomenon and delayed removal process are considered simultaneously. The removal processes in hard and complex faults of fault complexity model discussed in Sect. 2.2.3 is first obtained directly prior to the failure process definition by alternatively deriving the model in one stage. It can be noted here that models for hard and complex faults consider the time delay between failure observation and removal. This is done to first define the removal process in the presence of imperfect debugging and then obtain the failure process using Eq. (3.5.6).

The removal process for two stage SRGM [20] (observation and removal) considering learning and imperfect debugging is obtained using

$$b(t) = b\left[\frac{1}{1+\beta e^{-bt}} - \frac{1}{1+\beta+bt}\right] \tag{3.5.11}$$

The differential equation (3.5.3) with $a(t) = a + \alpha\, m_r(t)$ using above fault detection rate yields

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta+bt)e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \tag{3.5.12}$$

Similarly the removal process for three stage SRGM (observation, isolation and removal) considering learning and imperfect debugging is obtained using

$$b(t) = b\left[\frac{1}{1+\beta e^{-bt}} - \frac{(1+bt)}{(1+\beta+bt+(b^2t^2/2))}\right] \tag{3.5.13}$$

The differential equation (3.5.3) with $a(t) = a + \alpha\ m_r(t)$ using above fault detection rate yields

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta+bt+(b^2t^2/2))e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \tag{3.5.14}$$

These models can also be used to define the fault complexity. Superimposing the mean value functions (3.5.8), (3.5.12) and (3.5.14) we obtain an SRGM defining fault complexity, delayed removal process, effect imperfect debugging and learning phenomenon simultaneously.

3.6 Test Effort Based Imperfect Debugging Software Reliability Growth Models

In Chap. 2 we explained the importance of testing resource consideration in software reliability growth modeling. Several SRGM under perfect debugging environment were discussed there. An SRGM defined with respect to testing effort function, incorporating effect of imperfect debugging becomes more meaningful in terms of the useful information it can provide.

3.6.1 Pure Imperfect Fault Debugging Model

Kapur et al. [10] defined an s-shaped imperfect debugging model based on test efforts. The model is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) \Big/ w(t) = b(a - m_r(t)) - cp(t)(a - m_r(t)) \tag{3.6.1}$$

solution of the above model is given as

$$m_r(t) = a\left[\frac{(b-c)(1-e^{-b(W(t)-W(0))})}{(b-c)+ce^{-b(W(t)-W(0))}}\right] \tag{3.6.2}$$

It can be noted here that if $W(0) = 0$ and a Weibull type test effort function i.e. $W(t) = \alpha(1 - e(-\beta t^m))$ is used to describe the distribution of test effort, then we have $m_r(0) = 0$ and $m_r(\infty) = a[((b-c)(1-e^{-b\alpha}))/((b-c)+ce^{-b\alpha})]$ which implies that even if software, is tested for a long time, some fault will remain in the

software, which is contradictory to the imperfect debugging model with respect to time [13], which gives $m_r(\infty) = a$, i.e. all faults can be eliminated in infinite testing time. Such an optimistic forecast interferes the purpose of an imperfect debugging model. This analysis clearly illustrates the benefit of formulating imperfect debugging models with respect to test efforts. These models can depict more accurate utilization of test resources as well as reliability prediction at the time of software release.

3.6.2 Pure Error Generation Model

Huang et al. [21] studied the test effort based pure fault generation SRGM. They formulated Ohba and Chao [6] model with test efforts.

The differential equation for the model is described as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t)\Big/w(t) = b(a(t) - m(t)) \tag{3.6.3}$$

Using the fault content function $a(t) = a + \alpha m_r(t)$, mean value function of the SRGM is defined as

$$m_r(t) = (a/(1-\alpha))(1 - e^{-b(1-\alpha)(W(t)-W(0))}) \tag{3.6.4}$$

3.6.3 Integrated Imperfect Debugging Models

Kapur et al. [1] generalized integrated testing efficiency model proved to be very useful in terms of analyzing various existing perfect debugging and new models under imperfect debugging environment. Looking at the usefulness of this generalization, Kapur et al. [22] reformulated the model based on testing efforts to obtain more accurate and useful results. The generalized differential equation describing the failure and removal phenomenon are given by

$$\frac{\mathrm{d}}{\mathrm{d}t}m_f(t)\Big/w(t) = pb(t)[a(t) - m_r(t)] \tag{3.6.5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}a(t) = \alpha(t) \cdot \frac{\mathrm{d}}{\mathrm{d}t}m_r(t) \tag{3.6.6}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_f(t)\Big/w(t) = b(t)[a(t) - pm_f(t)] \tag{3.6.7}$$

Here again Eqs. (3.6.3) and (3.6.5) establishes the relationship

$$m_r(t) = pm_f(t) \tag{3.6.8}$$

Again defining the different forms of fault detection and removal rates per faults and fault content functions, different SRGM are derived with respect to the test effort function. This is left as an exercise for the readers to obtain test effort based models corresponding to the models discussed in Sect. 3.5.2.

The generalization (3.6.5) cannot be used to obtain integrated test effort based imperfect debugging SRGM considering learning phenomenon. In this case the fault detection/removal rate per additional fault needs to be redefined. Learning phenomenon of testing time is directly related to test efforts. Use of sound engineering principles, sophisticated tools and techniques, etc. can bring more learning. It calls for defining the learning function as a function of test efforts. Kapur et al. [22] carried a separate analysis for incorporating learning in test effort based imperfect debugging SRGM. They described the differential equation for the SRGM as

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t} = \frac{\mathrm{d}m(t)}{\mathrm{d}W(t)}\frac{\mathrm{d}W(t)}{\mathrm{d}t} \tag{3.6.9}$$

which can be defined as "the failure intensity during testing time t is the product of the failure intensity with respect to the test efforts and instantaneous test effort rate". Eq. (3.6.9) is expanded as

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t} = pb(W(t))(a(t) - m(t))\frac{\mathrm{d}W(t)}{\mathrm{d}t} \tag{3.6.10}$$

where

$$b(W(t)) = \frac{b}{1 + \beta e^{-bW(t)}} \quad \text{and} \quad a(t) = (a + \alpha m(t)) \tag{3.6.11}$$

Solving above equation under the initial condition $m(0) = 0$ and $W(0) = 0$ we get

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta)e^{-bW(t)}}{1 + \beta e^{-bW(t)}}\right)^{p(1-\alpha)}\right] \tag{3.6.12}$$

The mean value function of cumulated number of failures up to time t can be computed from the relationship $m_f(t) = m_r(t)/p$.

3.7 Reliability Analysis Under Imperfect Debugging Environment During Field Use

The reliability growth models discussed up to now are all applicable to the testing phase. We discussed the necessity of analyzing the operational reliability growth separately from the testing phase reliability growth in the previous chapters. The testing environment is usually very different from the operational environment.

In the testing phase, detection of a failure on the execution of a test case means a success, since the testing is aimed at detecting and correcting most of the faults lying in the software prior to release. But we know that software cannot be tested exhaustively before release due to constraints on time and cost. After a certain period of testing it is observed that any attempt to increase the reliability, results in exponential delay in the software release time and increase in cost. Hence testing is terminated at the time point when desired level of reliability is achieved as determined by some optimization routine, etc. and software is released for operational use. This is the reason, why we often hear about failures in operational-phase. The situation that occurs on detection of failures in the operational phase is not same as that in testing phase. The debugging activities cannot be started immediately. During field use, detection of a failure is a burden for the developers in terms of cost as well as goodwill loses. If a failure occurs in user environment, it is reported to the developer. The developer decides the amount of time and effort to be spent on removing the reported fault and testing the corrected code depending on the criticality and the urgency of the reported fault and agreement with the user, etc. [23]. Depending on the established strategy, a trouble ticket is created and assigned to some debuggers for analysis and code modification (i.e. fault removal) if the removal is not deferred. Firstly the code is tested for that failure and the fault is isolated and then removed. A perfect removal of fault in operational phase means reliability growth even during the field use. On account of the above discussion the SRGM developed for testing phase cannot be used as it is to predict the reliability growth during field use.

Now it is not unusual for a fault to occur multiple times in the field before it can be removed perfectly. Along with it some new faults can also be generated during these removals. Pham [24] mentioned that 22% of the faults are detected and 5% faults are generated during the operational phase. This implies that effect of both types of imperfect debugging may exist even during the field use. We have no doubt, as in case of testing phase, that selection of a perfect debugging SRGM for reliability estimation in operational phase will yield optimistic estimate. Hence an SRGM which incorporates the effect of testing efficiency should be a more judicious choice for practical applications even during field use. Only few imperfect debugging SRGM have been proposed in literature for operational reliability analysis.

3.7.1 A Pure Imperfect Fault Repair Model for Operational Phase

Jeske et al. [25] analyzed operation reliability growth relating probability of perfect debugging p, to the average time to remove a detected fault perfectly. They assumed that the expected time to remove a fault is μ times unit, i.e. $n\mu$ system time unit (assuming there are n systems in the field). The expected number of occurrences before it can be removed perfectly is $n\mu b$, therefore

$$\frac{1}{p} = 1 + n\mu b \tag{3.7.1}$$

substituting (3.7.1) in (3.3.4) we obtain the mean value function of the pure imperfect fault repair model, i.e.

$$m_f(t) = \left(\frac{a}{p}\right)(1 - e^{-bpt}) = a(1 + n\mu b)(1 - e^{-b\left(\frac{1}{1+n\mu b}\right)t}) \tag{3.7.2}$$

The authors claimed that the parameters a, b can be estimated from the failure data, while an estimate of μ is often known through experience with previous releases or some other software product. To account the mismatch between testing and operational environment, the estimate of b for the operational phase can be calculated using calibration factor approach, as the system test based estimate of b is usually very large due to the reason the test environment is set to expose maximum faults in shortest times.

The authors also proposed to account the reliability growth, accounting the fixed deferral of removal during operational phase. They distinguished the fault into two categories, first- faults are those removed from the current release, *Type-F*, contributing to reliability growth and second, fault whose removals are deferred to the future releases, *Type-D*, contributing to a constant component of overall software failure rate. The overall software failure rate is thus the sum of a decreasing and a constant failure rate. With software maturity in the field, the constant component will begin to dominate. On the other hand if all observed fault are removed in the field use, constant component would be zero. If λ_0 is the constant failure rate due to type-D faults, the failure intensity during operational phase would be given as

$$\lambda(t) = \lambda_0 + m_f'(t) = \lambda_0 + \lambda_F(t) = abe^{\left(-b\left(\frac{1}{1+n\mu b}\right)t\right)} \tag{3.7.3}$$

The parameters μ, b, and a of $\lambda_F(t)$ are the one associated with type-F faults.

3.7.2 An Integrated Imperfect Debugging SRGM for Operational Phase

Kapur et al. [22] extended their integrated testing efficiency model for testing phase (Eqs. (3.6.8) and (3.6.12)) to the operational phase replacing the test effort function by the usage function. Assuming

1. The number of failures during operation phase is dependent upon the usage function.
2. Usage function is a function of time and depends on the number of executions of the software in field.

To define appropriate usage functions, they classified the software into two categories namely

- *Project type software, and*
- *Product type software*

as already described in Chap. 2. During the testing phase, the type of software under testing does not affect the reliability growth since in this phase the testing environment or resource consumption does not depend on the software type. However, the type of software under consideration influences the reliability growth during the operational phase. More often some functions (major functionality of the software) are executed more frequently than others in the project type software. However the usage is limited to a particular user environment. On the other hand product type software are owned by a number of users executing it under different user environments. One user of product type software may use one function more frequently while other may use some other function, mostly depending on their need. These differences in the usage of the two types of software generate the need of defining distinct usage function for each type. *However in literature no distinction is made between the two types of software*. The models due to Kenny [24] and Kapur et al. [25] address the product type software (see Sect. 2.4)

3.7.2.1 Usage Function for Project Type Software

Project type software is owned by a specific organization for their specific use. In an organization many users may be accessing it either at a single location or at different locations. An exponential function is proposed to model the usage function for such software.

$$W(t) = r + s(1 - e^{-ct}) \tag{3.7.4}$$

where r represent the initial usage of the software when it is implemented in the user environment. As the time progresses the usage of software grows within the organization until it reaches the saturation level (time of update) $r + s$. Although some other functional form can also be used depending upon the user environment, number of people accessing the software and the usage of the software at each terminal, etc.

3.7.2.2 Usage Function for Product Type Software

Product type software is developed for a specific application according to the need of the software consumers in the market. Many different customers may buy a licensed copy of the software. One licensed copy of the software might be used by many users. For example an educational institution may buy software and many students' and\or faculty members might be using it. Number of executions of

product type software depends on the total number of user of the software and their usage intensity. Although commercial software products are in the market since years, identifying the target customers with certainty is impossible. Product software comes in the category of technological products and as such behaves as a new product or an innovative product when released in the market.

Kapur et al. [25] used Bass model of innovation diffusion [26] in marketing, for predicting the successive growth in the number of adopters of the dynamic market of software products over its life cycle. The model can adequately describe the users growth in terms of the factors stated above but it is slightly modified to describe the software usage appropriately. Adopters (or users) of software report a failure caused by some fault remaining in the software, to the developer. Once the number of users of the software is known, the rate at which the software is executed can be estimated.

For applying the Bass model it is assumed that there exists a finite population of prospective users who with time increasingly become actual users of the software (no distinction is made between users and purchasers as the Bass model has been successfully applied to describe growth in the number of both of them). In each period there will be both innovators and imitators using the software product. However as the process continues, the relative number of innovators will diminish monotonically with time. Imitators are however influenced by the number of previous buyers and increase relative to the number of innovators as the process continues.

The Bass model in terms of adoption distribution is given as

$$F(t) = \frac{1 - e^{-(p+q)t}}{1 + (q/p)e^{-(p+q)t}} \tag{3.7.5}$$

Shape of the resulting sales curve depends on the relative rate of these two sections of adopters—innovators and imitators. If software product is successful, the coefficient of imitation is likely to exceed the coefficient of innovation i.e. $p < q$. On the other hand, if $p > q$, the sales curve will fall continuously.

If $\bar{N}$ denotes upper limit of the number of licensed buyers of the software and γ is the average number of users within each user environment, then the total number of users of the software by time t is given as

$$S(t) = \bar{N}\gamma F(t) = mF(t) \quad \text{where } m = \bar{N}\gamma \tag{3.7.6}$$

Givon et al. [27] have used the modified version of the above model to estimate the number of licensed users as well as users of pirated copies of the software. It can be reasonably assumed that only the licensed-copy holders would report the failures, and hence Eq. (3.7.6) can be used to find the expected number of users at any time during the life cycle of the software. If the new software is expected to go through the same history as some previous software (very likely for different versions of the same software) the parameters of earlier growth curve may be used as an approximation.

Using the expression for expected number of licensed users of software the rate at which instructions in the software are executed can be estimated. Since the usage function depends on the number of executions of the software, therefore it is assumed that the usage function for product type software is a function of the total number of users of the software. For simplicity, v, a constant average execution rate is assumed for software usage within a user environment i.e.

$$W(t) = f(S(t)) = vS(t) \tag{3.7.7}$$

Some other functional relationship can be used depending upon the user environment and number of people assessing the software within a particular licensed user environment.

3.8 Data Analysis and Parameter Estimation

The model discussed in this chapter describes the relationship between the failure observation and/or fault removal process with respect to time or testing effort expenditure in an imperfect debugging environment. For the pure error generation models, when the delay in the removal process is assumed to be negligible the failure process describes the removal process whereas for the pure imperfect fault debugging models as well as integrated testing efficiency models, the number of failures is more than the number of removals. For pure error generation models an estimate of the detection process provides the estimate of the removal process. On the other hand for the pure imperfect fault debugging models and integrated testing efficiency models, first we fit the model equation for the detection process on the collected failure data and using these estimates we estimate the expected number of removals.

3.8.1 Application of Time Dependent SRGM

Failure Data Set

This data set is from Brooks and Motley [28]. The failure data set is for a radar system of size 124 KLOC (Kilo Lines of Code) tested for 35 weeks in which 1,301 faults were detected.

The following models have been chosen for data analysis and parameter estimation.

Model 1 (M1) Pure Imperfect Fault Debugging Model [4]

$$m_r(t) = a(1 - e^{-bpt}) \quad \text{and} \quad m_f(t) = (a/p)(1 - e^{-bpt})$$

Model 2 (M2) Pure Error Generation Model [6]

$$m(t) = (a/1-\alpha)(1-e^{-b(1-\alpha)t})$$

Model 3 (M3) Pure Error Generation Model [9]

$$m(t) = \frac{ab}{\alpha+b}(e^{\alpha t} - e^{-bt})$$

Model 4 (M4) Pure Error Generation Model [9]

$$m(t) = a(1-e^{-bt})\left(1-\frac{\alpha}{b}\right) + \alpha a t$$

Model 5 (M5) General Imperfect-Software-Debugging Model with S-Shaped FDR (PNZ model) [14]

$$m(t) = \frac{a(0)(1+\beta)}{1+\beta e^{-b(0)(1+\beta)t}}\left(\left(1-\frac{\alpha}{b(0)(1+\beta)}\right)(1-e^{-b(0)(1+\beta)t}) + \alpha t\right)$$

Model 6 (M6) PNZ model with Alternative Fault Content Function Pham [15]

$$m(t) = \frac{1}{1+\beta e^{-bt}}\left((k+a)(1-e^{-bt}) - \frac{a}{b-\alpha}(e^{-\alpha t} - e^{-bt})\right)$$

Model 7 (M7) Delayed S-shaped Pure Error Generation Model [16]

$$m(t) = \left(\frac{a}{1-\alpha}\right)\left(1-(1+bt)^{1-\alpha}e^{-b(1-\alpha)t}\right)$$

Model 8 (M8) Testing Efficiency Model [8]

$$m(t) = \frac{a}{p-\alpha}\left(1-\left(\frac{(1+\beta)e^{-bt}}{1+\beta e^{-bt}}\right)^{(c/b)(p-\alpha)}\right)$$

Model 9 (M9) Integrated Exponential Testing Efficiency Model [19]

$$m_r(t) = (a/1-\alpha)(1-e^{-bp(1-\alpha)t}) \quad \text{and} \quad pm_f(t) = m_r(t)$$

Model 10 (M10) Integrated Flexible Testing Efficiency Model Kapur et al. [1]

$$m_r(t) = \frac{a}{1-\alpha}\left[1-\left(\frac{(1+\beta)e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \quad \text{and} \quad pm_f(t) = m_r(t)$$

Model 11 (M11) Integrated S-shaped Testing Efficiency Model [20]

$$m_r(t) = \frac{a}{1-\alpha}[1-(1+bt)^{p(1-\alpha)}e^{-bp(1-\alpha)t}] \quad \text{and} \quad pm_f(t) = m_r(t)$$

Model 12 (M12) Integrated Flexible Testing Efficiency Model with Different FDR [20]

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta+bt)e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \quad \text{and} \quad pm_f(t) = m_r(t)$$

Model 13 (M13) Another Integrated Flexible Testing Efficiency Model with Different FDR [20]

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta+bt+(b^2t^2/2))e^{-bt}}{1+\beta e^{-bt}}\right)^{p(1-\alpha)}\right] \quad \text{and} \quad pm_f(t) = m_r(t)$$

The unknown parameters of all these models have been estimated using the regression module of SPSS. The values of estimated parameters have been tabulated in Table 3.1. Figures 3.2 and 3.3 show the goodness of fit curves for the estimation results tabulated in Table 3.1 and future predictions for exponential and s-shaped or flexile SRGM respectively.

The graphical plot of observed actual data depicts the s-shaped growth in the data. The models chosen for data analysis are exponential as well as s-shaped or flexible models. Models M1–M4 and M9 are all exponential. The data analysis results support the fact that an exponential model cannot be chosen to describe the testing process and reliability growth for this data set. The calculated value of mean square errors is very high for these models as compared to the s-shaped and flexible models. On the other hand models M5–M7 are flexible or s-shaped, but MSE value for these models is also high. The common characteristic of these models is that all of them are pure error generation models. This result suggests that pure error generation models are also not preferable for the analysis of this

Table 3.1 Estimation result for model 1 to model 13

Model	Estimated parameters						Comparison criteria	
	a	b	p	α	β	c,k	MSE	R^2
M1	8,155	0.0055	0.9000	–	–	–	9,766.90	0.9580
M2	9,201	0.0049	–	0.0500	–	–	9,762.67	0.9580
M3	2,128	0.0217	–	0.0125	–	–	10,218.82	0.9650
M4	1,248	0.0351	–	0.0317	–	–	9,916.57	0.9570
M5	919	0.0170	–	0.0000	1.61	–	8,969.44	0.9640
M6	1,976	0.9120	–	0.0370	78.00	34.12	6,430.64	0.9820
M7	1,689	0.0899	–	0.0001	–	–	2,967.68	0.9870
M8	1,239	0.1650	0.9580	0.0030	30.24	0.38	145.75	0.9990
M9	7,938	0.0060	0.9000	0.0790	–	–	12,600.62	0.9580
M10	1,290	0.2030	0.9600	0.0230	19.91	–	249.53	0.9990
M11	1,596	0.0931	0.9745	0.0363	–	–	3,116.99	0.9870
M12	1,325	0.1760	0.9970	0.0250	7.43	–	764.06	0.9970
M13	1,361	0.1840	0.9890	0.0200	1.41	–	1,290.49	0.9950

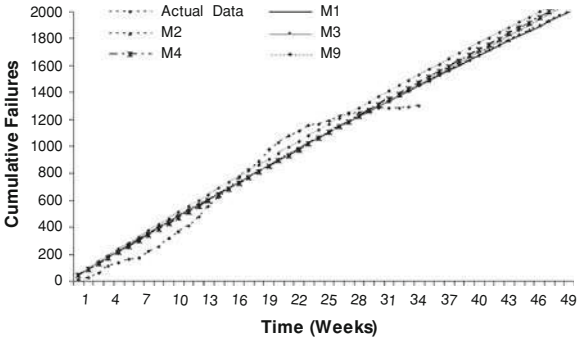


Fig. 3.2 Goodness of fit curve for model exponential models (M1–M4 and M9)

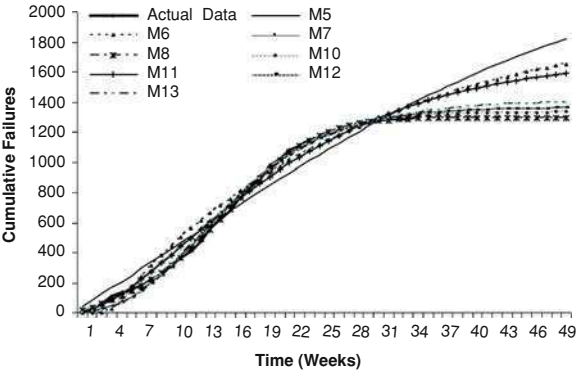


Fig. 3.3 Goodness of fit curve for s-shaped and flexible models (M5–M8 and M10–M13)

data set. In general the integrated testing efficiency models M8–M13, excluding M9 (due to exponential nature) and M11 has fitted well for the actual data. Model M8 seems to be the best fit model for the data set. However due to the formulation inconsistency in the model it cannot be chosen for estimating the failure and removal phenomena and predicting the system reliability. The next best fit model is M10 (Kapur et al. [1] integrated testing efficiency model) with MSE value close to M8 and similar R^2 values. Our analysis suggest that these models can be chosen for predicting the future failure/removal phenomena and reliability for the software project whose data set is been taken for the analysis.

The estimation result of the models M10 depict that the testing efficiency is 96% and the error generation rate is 2.3%. The value 19.91 for the shape parameter β implies the high s-shapedness of the actual data due to the fact that initially the failure observation phenomena picked very fast in the beginning then it increased at a low rate and later the rate of increase started increasing at a higher rate. It is estimated that a total of 1,308 faults are observed in the 35 months time and out of the1,308 observed faults 1,295 were removed successfully in the same time period. Further predictive ability of the model can be established by conducting the predictive validity test. Establishing predictive ability of this model is left as an exercise for the readers.

3.8.2 An Application for Integrated Test Effort Based Testing Efficiency SRGM

We have seen in the above analysis that flexible integrated testing efficiency has provided that best fit on the data set taken for the analysis. This result is true in general as whether the observed data is exponential or s-shaped, an appropriate growth curve is captured by data analysis due to the presence of shape parameter (β in this case). This parameter takes a value either zero or approximately zero when shape of the growth of collected data is exponential or otherwise it takes a positive value with magnitude according to the s-shapedness of the growth curve. On the other hand the magnitude of testing efficiency is captured by the testing efficiency parameters. For a highly efficient testing, efficiency the parameter p takes value near to 1 and α takes value near to zero and vice versa. Hence all types of situations are well captured by flexible testing efficiency models. Now we show the parameter estimation and model validation of a flexible integrated testing efficiency SRGM with respect to the testing efforts.

Failure Data Set

This failure data set is for a command, control and communication system cited in Brooks and Motley [30]. The software was tested for 12 months and 2,657 faults were identified during this period. The following model is taken for the analysis

Model 14 (M14) Integrated Imperfect Debugging Test Effort based Model [22]

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta)e^{-bW(t)}}{1+\beta e^{-bW(t)}}\right)^{p(1-\alpha)}\right]$$

First we have fitted four test effort functions exponential, Rayleigh, Weibull and logistic on the observed test effort data and then using the best fit function the parameters of the SRGM are estimated. The estimation results with the comparison criteria values of test effort function are given in Table 3.2. The values of estimated parameters for the SRGM have been tabulated in Table 3.3. The

Table 3.2 Estimation results for testing effort functions

Test effort function	Estimated parameters			Comparison criteria	
	α	v	c,β	MSE	R^2
Exponential	35,237	0.0297	–	15,443.51	0.999
Rayleigh	10,153	0.0491	–	497,788.67	0.953
Weibull	11,714	0.3488	8.773	34,637.04	0.997
Logistic	33,524	0.0313	1.002	17,426.71	0.998

Table 3.3 Estimation results of SRGM for testing phase with exponential effort function

Estimated parameters model 14					Comparison criteria	
a	B	β	p	α	MSE	R^2
3,322	0.000141	0.06468	0.9020	0.000115	1,615.22	0.998

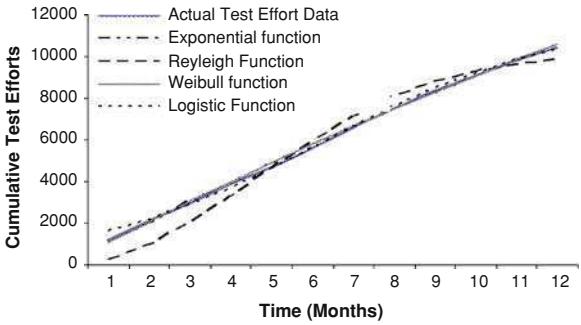


Fig. 3.4 Goodness of fit curve for test effort functions

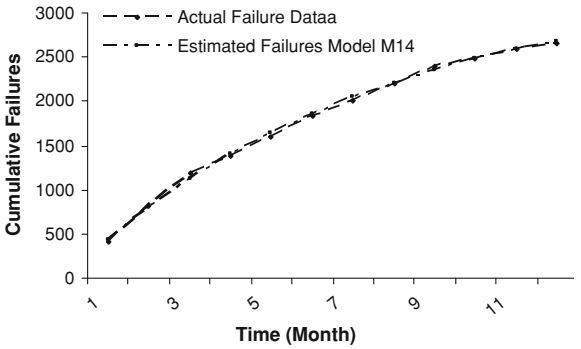


Fig. 3.5 Goodness of fit curve for test effort based model M14

goodness of fit curves for the estimation results of test effort functions and SRGM are shown in Figs. 3.4 and 3.5 respectively.

3.8.3 An Application for Integrated Operational Phase Testing Efficiency SRGM

In this chapter we have also discussed integrated testing efficiency SRGM applicable for predicting software failure and fault removal phenomena and measuring the reliability for the operational phase. Two types of SRGM have been discussed, one developed for the project type software and the other for the product type software. Now we establish the validity of these SRGM.

3.8.3.1 Data Analysis of SRGM for Project Type Software

Failure Data Set

This failure data is for Real time system collected for operational phase cited in Musa [31]. This is an interval domain data available for 192 days during which 37 faults were identified.

Model 15 (M15) In model 14 the test effort function is replaced by a usage function. Usage Function for Project Type Software

$$W(t) = r + s(1 - e^{-ct})$$

Substituting the usage function in the model, equation parameters of the mean value function for the SRGM in operational phase for project type software are estimated. The estimation results are tabulated in Table 3.4. The Fitting of the model is illustrated graphically in Fig. 3.6. Further the data is truncated into different proportions and is used to estimate the parameters of the SRGM. For each truncation, we have computed the relative prediction error. The Predictive validity test results are shown in Table 3.5. Figure 3.7 shows the graphical plot of relative prediction error.

It is observed that the predictive validity of the model varies from one truncation to another. Even 85% of data is sufficient to predict the future failure behavior well with RPE as low as −10.21% for this SRGM.

Table 3.4 Estimation results of SRGM for operational phase for project type software

Estimated parameters model M15								Comparison criteria	
a	b	β	p	α	r	s	c	MSE	R^2
37	0.04575	71.267	0.9	0.001	0.1	156	0.01361	2.59	0.9880

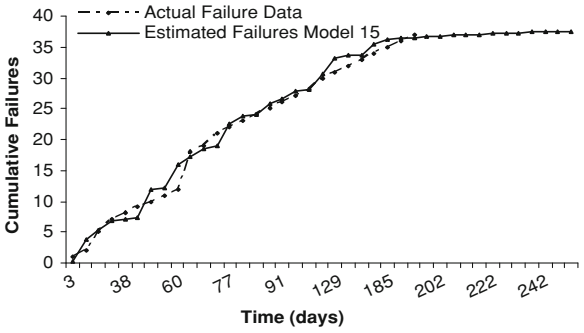


Fig. 3.6 Goodness of fit curve for model M15

Table 3.5 Predictive analysis results for model M15

Data truncation (%)	m_r (192)	RPE	RPE (%)
100	36.53	−0.0127	−1.27
95	34.69	−0.0624	−6.24
90	33.66	−0.0903	−9.03
85	33.22	−0.1021	−10.21
80	32.82	−0.1131	−11.31
70	32.82	−0.1131	−11.31
60	31.64	−0.1447	−14.47

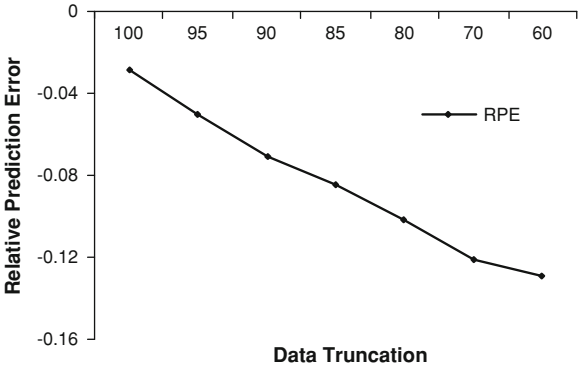


Fig. 3.7 Relative prediction error for the predictive validity test of model M15

3.8.3.2 Data Analysis of SRGM for Product Type Software

Failure Data Set

This failure data is for an operating system collected for operational phase cited in Musa [31]. The interval domain data is available for 148 day during which 112 faults were identified.

Model 16 (M16) Usage Function for Product Type Software

$$W(t) = vm\frac{1 - e^{-(p+q)t}}{1 + (q/p)e^{-(p+q)t}}$$

Substituting the usage function in the model, equation parameters of the mean value function for the SRGM in operational phase for product type software are

Table 3.6 Estimation results of SRGM in operational phase for product type software

Estimated parameters model M15									Comparison criteria	
a	b	β	p	α	m	r	s	v	MSE	R^2
142.88	0.0294	7.218	0.9537	0.0466	8.28	0.0081	0.0064	16.5	25.14	0.9925

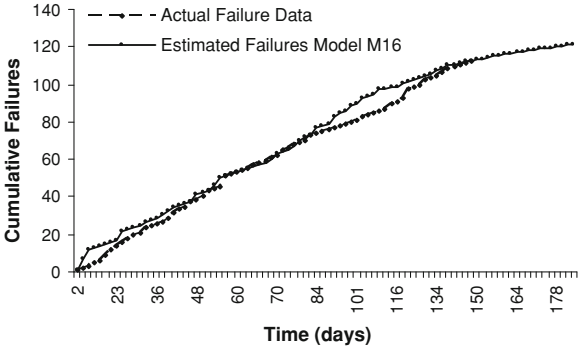


Fig. 3.8 Goodness of fit curve for model M16

Table 3.7 Predictive analysis results for model M16

Data truncation (%)	m_r (148)	RPE	RPE (%)
100	112.60	0.0053	0.535
95	106.52	−0.0490	−4.896
90	105.41	−0.0588	−5.882
85	97.48	−0.1297	−12.968
80	93.92	−0.1614	−16.143
70	89.94	−0.1970	−19.699
60	91.64	−0.1818	−18.179

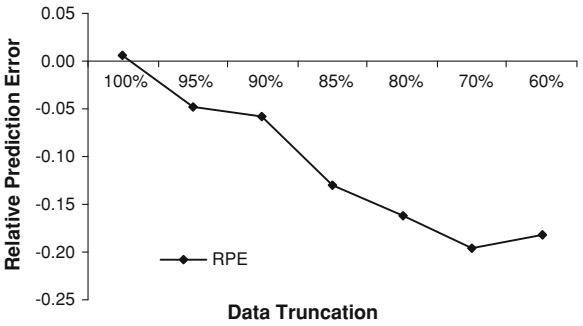


Fig. 3.9 Relative prediction error for the predictive validity test of model M16

estimated. The estimation results are tabulated in Table 3.6. The Fitting of the model is illustrated graphically in Fig. 3.8. Further we have carriedout predictive validity analysis on the collected data, truncating collected data into different proportions. For each truncation the relative prediction error is computed. The results are tabulated in Table 3.7. Figure 3.9 shows the graphical plot of relative prediction error.

Predictive validity results implies that at least 90% of observed data is required to predict the future failure behavior well with RPE –5.882% for this model on this data. For 85% of data, the RPE is –12.97%.

Exercises

1. What is imperfect debugging? What types of imperfect debugging can be occur while testing the software?
2. A test case is executed during the testing of software in the testing phase. The execution of test case resulted in a failure, indicating the presence of some fault on some path of the module which is executed on the test run. The software debuggers identified the fault and attempted to correct it. What would be the fault count of the software after the fault correction attempt?
3. The failure phenomenon of a pure error generation type SRGM is described as

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t} = b(t)[a(t) - m(t)]$$

   If the failure rate $b(t) = b$ and $a(t) = a + K(1 - e^{-\theta t})$, obtain the mean value function of the SRGM.
4. The mean value function of the removal phenomenon of integrated imperfect debugging SRGM is given by

$$m_r(t) = e^{-p\int (1-\alpha(u))b(u)\,du}\left[ap\int_0^t b(x)e^{-p\int (1-\alpha(x))b(x)\,dx}\,dx\right]$$

   If $b(t) = \left(\frac{b^2 t}{1+bt}\right)$ and $\alpha(t) = \alpha$ then

$$m_r(t) = \frac{a}{1-\alpha}[1 - (1+bt)^{p(1-\alpha)}e^{-bp(1-\alpha)t}]$$

5. Show that $m_r(t) = pm_f(t)$ if $\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = pb(t)[a(t) - m_r(t)]$ and $\frac{\mathrm{d}}{\mathrm{d}t}m_f(t) = b(t)[a(t) - pm_f(t)]$.
6. Obtain the mean value function of the exponential test effort based integrated imperfect debugging SRGM if the fault content function is given as $a(t) = a + \alpha m_r(t)$.
7. A model developed to describe the reliability growth of software during testing phase, fails in adequately describing the reliability growth during the field usage. Comment.
8. Suppose the usage function of project type software can also be described by $W(t) = r + st^k$, where r, s and k are constant. Determine the unknown parameters of the SRGM for project type software in operational phase using usage function specified above and compare the result on mean square error with the results of Sect. 3.8.3.

References

1. Kapur PK, Kumar D, Gupta A, Jha PC (2006) On how to model software reliability growth in the presence of imperfect debugging and fault generation. In: Proceedings 2nd international conference on reliability and safety engineering, INCRESE, pp 261–268
2. Goel AL (1985) Software reliability models: assumptions, limitations and applicability. IEEE Trans Software Eng SE-11:1411–1423
3. Jelinski Z, Moranda P (1972) Software reliability research. In: Freiberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 465–484
4. Kapur PK, Garg RB (1990) Optimal software release policies for software reliability growth models under imperfect debugging. Recherché Operationanelle/Operations Research 24:295–305
5. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliability R-28(3):206–211
6. Ohba M, Chou X M (1989) Does imperfect debugging effect software reliability growth. In: Proceedings 11th international conference of software engineering, pp 237–244
7. Xie M (2003) A study of the effect of imperfect debugging on software development cost. IEEE Trans Software Eng 29(5):471–473
8. Zhang X, Teng X, Pham H (2003) Considering fault removal efficiency in software reliability assessment. IEEE Trans Syst Man Cybern Part A Syst Humans 33(1):114–120
9. Yamada S, Tokunou K, Osaki S (1992) Imperfect debugging models with fault introduction rate for software reliability assessment. Int J Syst Sci 23(12):2253–2264
10. Kapur PK, Grover PS, Younes S (1994) Modeling an imperfect debugging phenomenon with testing effort. In: Proceedings 5th international symposium on software reliability engineering (ISSRE'1994). IEEE Computer Society, Monterey, pp 178–183
11. Lynch T, Pham H, Kuo W (1994) Modeling software-reliability with multiple failure-types and imperfect debugging. In: 1994 proceedings annual reliability and maintainability symposium, pp 235–240
12. Lo HJ, Huang CY (2004) Incorporating imperfect debugging into software fault processes. TENCON 2004. In: 2004 IEEE region 10 conference, vol 2, 21–24 November 2004, pp 326–329
13. Kapur PK, Younes S (1996) Modeling an imperfect debugging phenomenon in software reliability. Microelectron Reliability 36(5):645–650
14. Pham H, Nordmann L, Zhang XA (1999) General imperfect-software-debugging model with s-shaped fault-detection rate. IEEE Trans Reliability 48(2):169–175
15. Pham H (2000) Software reliability. Springer-Verlag, New York
16. Kapur P K, Singh O, Gupta A (2005e) Some modeling peculiarities in software reliability. Quality, reliability and infocom technology, trends and future directions. Narosa Publications Pt. Ltd., New Delhi, pp 20–34
17. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliability R-32(5):475–484
18. Pham H, Zhang X (2003) NHPP software reliability and cost models with testing coverage. Eur J Oper Res 145(2):443–454
19. Kapur PK, Gupta A, Jha PC (2007) Reliability growth modeling and optimal release policy of a n-version programming system incorporating the effect of fault removal efficiency. Int J Autom Comput 4(4):369–379
20. Kumar D, Kapur R, Sehgal VK, Jha PC (2007) On the development of software reliability growth models with two types of imperfect debugging. Commun Dependability Qual Manag Int J 10(3):105–122
21. Huang CY, Kuo SY, Lyu MR, Lo HJ (2000) Effort-index-based software reliability growth models and performance assessment. In: Proceedings 24th annual international computer software and applications conference (COMPSAC 2000), Taipei, Taiwan, 25–27 October 2000, pp 454–459

22. Kapur PK, Gupta A, Jha PC (2007) Reliability analysis of project and product type software in operational phase incorporating the effect of fault removal efficiency. Int J Reliability Qual Safety Eng 14(3):219–240
23. ESA Board for Software Standardization and Control (1995) Guide to the software operations and maintenance phase. European Space Agency, 8-10, rue Mario-Nikis, 75738 PARIS CEDEX, France, ESA PSS-05-07 Issue 1 Revision 1
24. Pham H (2006) System software reliability. Reliability engineering series. Springer, London
25. Jeske DR, Zhang X, Pham L (2001) Accounting for realities when estimating the field failure rate of software. In: Proceedings 12th international symposium on software reliability engineering, pp 332–339
26. Kenny GQ (1993) Estimating defects in a commercial software during operational use. IEEE Trans Reliability 42(1):107–115
27. Kapur PK, Bardhan AK, Jha PC (2003) Optimal reliability allocation problem for a modular software system. OPSEARCH J Oper Res Soc India 40(2):133–148
28. Bass FM (1969) A new product growth model for consumer durables. Manag Sci 15:215–227
29. Givon M, Mahajan V, Muller E (1980) Software piracy: estimation of lost sales and the impact on software diffusion. J Marketing 59:29–37
30. Brooks WD, Motley RW (1980) Analysis of discrete software reliability models—technical report (RADC-TR-80-84). Rome Air Development Center, New York
31. Musa JD (1980) Software reliability data. Data and Analysis Center for software, USA, www.dacs.dtic.mil

Chapter 4
Testing-Coverage and Testing-Domain Models

4.1 Introduction

Software development is a very complex and dynamic process. Every phase of software development can be further divided into a number of sub-phases, where each sub-phase has its own contribution to the software development process. Lot many activities are involved in each phase/sub-phase. Detailed study of each activity of this development process requires building an understanding of the various associated philosophies, theories and concepts. Same is true for the software testing phase and quality measurement. Reliability is considered as the key characteristic of quality. Until now in the previous chapters we have seen that there are various aspects which need to be considered during reliability assessment. Along with those which have been discussed in the previous chapters, two characteristics of reliability assessment which have not been discussed so far are—*Testing Coverage* and *Testing Domain Ratio*. Both these characteristics play an important role in reliability estimation and decision related to the testing and operational phase. In this chapter, the concepts of testing coverage, testing domain ratio and software reliability growth modeling incorporating these measures have been discussed in detail.

4.1.1 An Introduction to Testing-Coverage

Planning software release schedule is a very crucial decision. Achieving a level of quality forms the main basis for deciding the release schedules. The level of testing required usually depends on the potential consequences of undetected bugs. Besides testing efficiency, testing efforts, various other facts are incorporated during reliability estimation such as fault complexity, debugging time lag etc. There are many other factors that greatly influence the reliability growth. Among

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_4,
© Springer-Verlag London Limited 2011

all others, one factor that plays a critical role in reliability assessment is *Testing Coverage*. There are various notions on the relationship between testing coverage and reliability estimation. The most relevant is related to the role of test coverage in developing more efficient test or in determining an effective and non-effective test. There are various test methodologies which can be adopted to test a software such as function testing, white box testing, data flow coverage, decision flow coverage, mutation testing etc. To each testing method is associated a saturation effect. An understanding of this saturation effect is key to the understanding of importance of test coverage in reliability estimation. Saturation effect is defined as the tendency of testing methods to limit their ability to expose faults in a program under testing. After reaching this limit, continuing testing adopting the same method may cause significant over or under estimation of reliability.

More precisely, most of the software reliability models used to assess the reliability are either time based or effort based. These models use the failure history obtained during testing to predict the field behavior of the software under test, under the assumption that testing is performed in accordance with the given operational profile. Two main difficulties are associated with the use of these models for practical application. First problem is related to the operational distribution. The test collection developed from a known operational profile may be different from the one that would occur in practice. Otherwise the software may be put to different operational profiles in field or it is also possible that the software is completely a new and no known operational profile is obtainable. Second, when software is tested using a collection of test cases, as the testing progresses, more faults are detected and removed. Testing may be continued using similar collection developed in accordance to some test methodology. If time between failures is the only consideration in the reliability estimation, then sever drawback that can result is over-estimation of reliability than actually achieved. Along with this, another fact that may cause over estimation is that the test cases generated/executed in the later phase of testing are less likely to cause the program to fail than those generated in the earlier phases developed for a known operational profile. This problem can be solved if we can identify the redundant test efforts i.e. one needs to determine which test case is redundant and how much test effort is actually desired for effective testing and accurate reliability estimation.

Conclusively we can say that a less effective test strategy may at many times prove to be less efficient in finding defects with the same amount of test efforts put with effective strategy. To ensure the quality and accurately estimate and predict the reliability of software, it is necessary that it should be tested until all the constructs in the programs achieve a desired level of coverage. In this context Gokhale et al. [1] gave a unifying definition for testing coverage which accommodates all constructs of the system that are to be covered by the testing. Given a software product and its companion test set, they defined testing coverage as "*the ratio of the number of potential fault sites sensitized by the test divided by the total number of potential fault sites under consideration*". Test coverage analysis is a structural testing technique that helps eliminate gaps in a test suite. It helps most in the absence of a detailed, up-to-date requirements specification. The importance of

investigating the effect of testing coverage for reliability measurement has been established by several researchers. Empirical evidence strongly suggests that Testing, which is carried out with some form of coverage measurement may fail to sensitize as much as 45% of the code [1]. More rigorous testing is desired in complex or more frequently called modules. The testing coverage measure can directly be related to the code coverage achieved during test case execution. *Testing Coverage* measure can assist in assuring the quality of test cases in such a way that nearly 100% code can be covered thereby assuring the quality of the software. Greater the level of code coverage higher the level of reliability achieved. Software designers and testers must develop effective evaluation tools capable of measuring test coverage and pointing out design deficiencies contributing to poor software testability. Such tools can also provide data which can lead to improvement in quality and effectiveness of a software test. Software reliability models that have been formulated based on testing time and coverage simultaneously can use this data for accurate reliability measurement.

Another importance of test coverage measure lies in the fact that it is important for both software developer's as well as users. It helps developers to evaluate the quality of the tested software, determine the additional testing desired to achieve the necessary reliability and adequately planning the release schedules. On the other hand, it is a quantitative confidence criterion for the customer in taking the decision to buy the product. After revealing the importance of testing coverage measure in reliability measurement, we now give some definitions, related concepts and methods of testing coverage measurement.

On the part of software testers, *Test Coverage Analysis* is the process of

- Finding areas of a program not covered by a set of test cases
- Creating additional test cases to increase coverage
- Determining a quantitative measure of test coverage
- Identifying redundant test cases that do not increase coverage

In general test coverage measure is defined as how well a test covers all the potential fault sites in the software under test. Potential fault site here mean program entities representing either structural or functional program elements whose sensitization is reckoned essential towards establishing the operational integrity of the software. A large variety of test coverage measures exist. Now we give a brief description of some well-known coverage measures [2].

4.1.1.1 Statement Coverage

This is the most simple coverage measure and a number of open source products exist that measure this level of coverage. It is defined as the fraction of the total number of statements that have been executed by the test data i.e. this measure reports whether each statement is executed or not. It is also known as line coverage, segment coverage, and basic block coverage. The chief advantage of this measure is that it can be applied directly to object code and does not require

processing source code. However the measure is discouraged as it does not identify bugs that arise from the control flow constructs in the source code, such as compound conditions or consecutive switch labels as it does not check whether all options in a branch have been covered and is completely insensitive to the logical operators. This means that even if we can easily get 100% coverage, we may encounter glaring, uncaught bugs.

4.1.1.2 Branch Coverage

The coverage, also called decision coverage, reports whether a test case has explored both true and false outcomes of Boolean expressions in control structures. It considers the entire Boolean expression as a single predicate, evaluating to true or false, ignoring the branches within the expression which may occur due to short-circuit logical operators. It contains coverage measures for switch-statement, exception handlers etc. that are not included in statement coverage. The basic metric of this measure is its simplicity without the problems of statement coverage. A disadvantage is that this metric ignores branches within Boolean expressions which occur due to short-circuit operators.

4.1.1.3 Condition Coverage

Condition coverage measures the sub-expressions occurring in a Boolean expression independently of each other and reports the *true* and *false* outcome for these, separated by logical-and and logical-or if they occur. It is similar to branch coverage but has better sensitivity to the control flow. However, full condition coverage does not guarantee full decision coverage. Another variant of this type is multiple condition coverage which reports whether every possible combination of Boolean sub-expressions occurs. The test cases required for full multiple condition coverage of a condition are given by the logical operator truth table for the condition, hence for this metric determining the minimum set of test cases required can be tedious, especially for very complex Boolean expressions. A hybrid measure of this type is also called condition/decision coverage metric composed by the union of condition coverage and decision coverage. It has the advantage of simplicity but without the shortcomings of its component metrics.

4.1.1.4 Path Coverage

This measure is used for the most thorough testing of software. The metric reports whether all possible paths in each function of the program have been followed. A path is defined as a unique sequence of branches from the entry point to the exit of a function. It is also known as predicate coverage, which views paths as possible combinations of logical conditions. A large number of variations of this metric

exists to cope with loops which may introduce unbounded number of paths, such as considering only a limited number of looping possibilities. Path coverage has two sever disadvantages. The first is that it needs a huge number of probes to monitor all the paths in a program, which are exponential to the number of branches. The second disadvantage is that many paths considered by it are impossible to exercise due to relationships of data.

4.1.1.5 Data Flow Coverage

This measure is a variation of path coverage and considers only the sub-paths from variable assignments to subsequent references of the variables. The advantage of this measure is that the paths reported are pertinent to the manner in which the program handles data. It also suffers from two disadvantages: the first is that it does not include decision coverage and another is that it is very complex which makes it difficult to implement.

4.1.1.6 Function Coverage

This metric report's whether each function or procedure has been invoked. It is useful during preliminary testing to assure at least some coverage in all areas of the software. Broad, shallow testing finds gross deficiencies in a test suite quickly.

4.1.1.7 Call Coverage

Also known as pair coverage, this metric reports whether each function call is executed. It is based on the hypothesis that bugs commonly occur in interfaces between modules.

There are many other coverage measures used in practice depending on the requirements of the testers such as object code branch coverage, loop coverage, race coverage, relational operator coverage, table coverage, weak mutation coverage, etc. Names of all these coverage measure's are reflective of how the metric computes the coverage.

4.1.2 An Introduction to Testing Domain

The concept of testing domain is closely related to testing coverage. We know the aim of testing coverage analysis is to quantitatively, define whether all the constructs in the programs have been covered or not by the test executed to test the software. Different types of coverage measures give measure of coverage with respect to the different aspects of the software such as statement, branch, path etc.

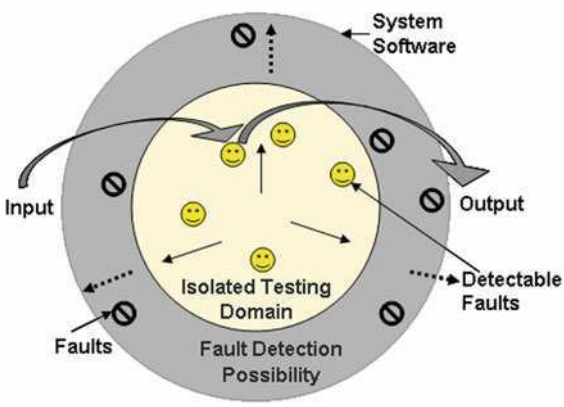


Fig. 4.1 A testing domain of a software system

On the other hand the testing domain is related to the domain of software influenced by the test cases. The test cases executed on the software in the testing phase are developed to influence the faults lying dormant in the various modules\functions implemented in the software based on the requirement specifications. These test cases indeed influence a set of testing paths of these modules and functions. This set of testing paths, all of which are to be eventually influenced by designed test cases is called the *testing domain* [3]. The domain of testing which gets influenced by the test cases executed by any time in the testing phase is called the *isolated testing domain* by that time. The isolated testing domain expands as the number of executed test cases increases which in turn makes more faults detectable by the debuggers. The goal of testing with respect to isolated testing domain is to expand it over the entire software system. Figure 4.1 illustrates a basic testing domain.

The isolated testing domain ratio in software is closely related to the quantity and quality of the executed test cases, which in turn is influenced by the testing skill of the test case designer. Increase in the number of detectable faults during testing is strongly influenced by the efficiency of the test case designers. Fujiwara and Yamada [4] defined two cases of testing skill related to testing domain and developed software reliability growth models with respect to isolated testing domain for both cases. Software reliability modeling with respect to the isolated testing domain enables the software developer to know the relation between the isolated testing domain ratio and the number of faults detectable by a testing method.

The efficiency of a test case design is related to the isolated testing domain as follows:

Case of low skill If the test designers are inexperienced or they are not pocessing high professional skills. The degree of comprehension of the internal structure by such test designers of the software is low. It results in deterioration in the quality of test case design and hence slow testing domain growth rate.

Case of high skill Expert test designers possess high degree of internal structure comprehension. Using their technical skills and experience, they develop high

quality test cases and an effective overall test. Their expertise results in increasing rate of testing domain growth rate and hence an increasing rate of the detectable faults.

The effect of testing skill on isolated testing domain is most remarkable in the early stages of testing. When the skill of test designers is low, it is most likely that only a narrow domain of testing can be isolated as the test cases may not be efficient in influencing the isolated testing domain. Such a test cannot necessarily detect many faults. On the other hand if the test developers have experience of similar projects in the past along with professional skills, then the test designed by them influence the testing domain at a fast rate and will isolate almost whole of the software paths. It will result in more number of detectable faults as well as higher reliability at the end of testing phase.

Only a few attempts have been made in the literature for modeling the testing domain growth rate, expected number of faults detectable in the isolated testing domain and software reliability estimation with respect to the expected number of detectable faults. Along with testing coverage based SRGM in this chapter we have discusses various software reliability models based on isolated testing domain.

4.2 Software Reliability Growth Modeling Based on Testing Coverage

Analysis of effectiveness of testing and test cases using the testing coverage measure started as early as early 1980s. The initial attempts were related to measuring coverage for random testing [5], test analysis using structural coverage [6] and examine fault exposure ratio [7, 8]. The initial attempt to relating testing coverage to software reliability seems to be made by Malaiya et al. [9] based on the previous work.

4.2.1 Relating Testing Coverage to Software Reliability: An Initial Study

In the introduction section we defined a number of coverage measures related to software testing. In general any testing coverage measure is measured in terms of structural or data-flow units or enumerables that have been exercised. Malaiya et al. [2, 5] considered five types of enumerables to formulate their model, namely—defect, statement, branch, C-use and P-use coverage. Some of them have already been defined previously. Now we define the remaining.

Defect Coverage the fraction of actual defects initially present that would be detected by a given test set measures the defect coverage.

C-use Coverage the fraction of total number of computational use (C-use) that has been covered by one C-use path during testing. A C-use path is a path through a program from each point where the value of a variable is modified to each C-use (without the variable being modified along the path).

P-use Coverage the fraction of total number of P-uses that have been covered by one P-use path during testing. A P-use path is a path from each point where the value of a variable is modified to each P-use, a use in a predictable or decision (without modifications to the variable along the path).

The coverage increases when more tests are applied, provided that the test cases are not repeated and complete test coverage has not already been achieved. A small number of enumerables may not be reachable in practice due to their infeasibility or very low testability. Authors assumed that the fraction of such enumerables is negligible. The achieved coverage by a set of tests depends not only on the number of tests applied (or equivalently, the testing time) but also on the distribution of testability values of the enumerables. A statement which is reached more easily is more testable i.e., is more likely to get covered with only a small number of tests. Along with this, testability also depends on the likelihood that a fault reached actually causes a failure. Enumerables that have less frequency of execution in the field use have low testability and may not get exercised by most of the tests. It is also observed that the distribution of testability values may shift with the testing progress. The easy testable enumerables gets covered mostly in the early testing while the hard once remain and gets covered in the later testing at a less rate. It implies that the rate of growth of testing coverage decreases with testing progress and reaches a negligible value when almost 100% coverage is achieved.

Notation

p^j_{di}	Fraction of all enumerables of type j having detectability di
$\alpha^j(x, x + dx)$	Enumerable of type j with detectability between x and $x + dx$
$C^j(n)$	Expected coverage of the enumerable of type j
m_1	Parameter of one parameter model
N^j	Total number of enumerables of type j
β^j_0, β^j_1	Parameters of logarithmic model
$K^0(0)$	Defect exposure ratio at time $t = 0$
K	Overall fault exposure ratio
T_L	The linear execution time
α^0	Parameter describing the variation in the exposure ratio

The model is based on the detectability values of the enumerables. Malaiya et al. [9] defined detectability (d^j_l) as the probability that the lth enumerable of type j will be exercised by the randomly chosen test and the detectability profile is the distribution of detectability values in the system under test. The authors proposed discrete Normalized Detectability Profile (NDP) for formulating their model. NDP for a system under test is defined by the vector

$$p^j = \{p^j_{d1}, p^j_{d2}, \ldots p^j_{di}, \ldots, p^j_{du}\}; d^j_{i-1} < d^j_i < d^j_{i+1} \tag{4.2.1}$$

where $\sum_{di=0}^{i} p^j_{di} = 1$.

Enumerables may have detectability value 0 as it may not be feasible or testable due to redundancy in implementation. A continuous function is approximated for NDP if enumerables are large.

The Continuous NDP for the system under test is defined as

$$p^j(x)\mathrm{d}x = \frac{N^j(x, x+\mathrm{d}x)}{N^j}; \quad 0 \le x \le 1 \text{ and } \int_0^1 p^j(x)\mathrm{d}x = 1 \tag{4.2.2}$$

4.2.1.1 One Parameter Model for Testing Coverage

Now assuming random testing method i.e. any single test can be selected randomly with replacement, the expected coverage of the enumerables of type j for discrete NDP is defined as

$$C^j(n) = 1 - \sum_{i=1}^{n} \left(1 - d^j_i\right)^n p^j_i \tag{4.2.3}$$

and for continuous NDP it is

$$C^j(n) = 1 - \int_0^1 (1-x)^n p(x)\mathrm{d}x \tag{4.2.4}$$

However the actual software testing is more likely to be pseudo-random, since usually any test once applied are not repeated. Random testing is hence a reasonable approximation is this case except when coverage approaches 100%. The Eqs. (4.2.3) and (4.2.4) requires the knowledge of exact detectability profiles, which requires lot of computation. An approximation for detectability profiles is given as follows.

When one test is applied, the probability that an enumerable with detectability d^j_i will not be covered is $(1 - d^j_i)$. The probability that an enumerable will not be covered by n tests and remains a part of the profile is $(1 - d^j_i)^n$. Thus if Eq. (4.2.1) gives the initial discrete NDP, after n tests it is given as

$$p^j_n = \left\{p^j_{d1}\left(1 - d^j_1\right)^n, p^j_{d2}\left(1 - d^j_2\right)^n, \ldots, p^j_{du}\left(1 - d^j_u\right)^n\right\} \tag{4.2.5}$$

Equivalently the continuous profile is given as

$$p^j_n(x) = p_n(x)(1-x)^n \tag{4.2.6}$$

The results suggests an initial detectability profile

$$p^j(x) = (m_j + 1)(1-x)^{m_j} \tag{4.2.7}$$

Factor $(m_j + 1)$ ensures that the area under the initial profile curve is unity. Substituting (4.2.7) in (4.2.4) implies

$$C^j(n) = 1 - (m_j + 1)\int_0^1 (1-x)^{m_j+n} \mathrm{d}x = \frac{n}{m_j + n + 1} \tag{4.2.8}$$

above is a one parameter model for testing coverage based on parameter m_j.

4.2.1.2 Logarithmic Coverage Model

Instead of random testing in actual practice, a test case is selected in order to exercise functionality or enumerable that is remaining untested. It makes testing more directed and efficient than random testing. The coverage growth depends on the detectability profile along with test selection strategy. Following the analysis of Malaiya et al. [7] that the defect coverage growth in practice can be described by a logarithmic model, it is assumed that the coverage growth of other enumerable types also be logarithmic. The following model is given

$$C^j(t) = \frac{1}{N^j}\beta_0^j \ln(1 + \beta_1^j t) \tag{4.2.9}$$

If single application of test takes T_s seconds then the time needed to apply n test is nT_s hence

$$C^j(n) = \frac{1}{N^j}\beta_0^j \ln(1 + \beta_1^j T_s n) \tag{4.2.10}$$

defining $b_0^j = \frac{\beta_0^j}{N^j}$ and $b_1^j = \beta_1^j T_s$ the coverage model can be rewritten as

$$C^j(n) = b_0^j \ln(1 + b_1^j n); \quad C^j(n) \le 1 \tag{4.2.11}$$

For defect coverage (C^0) i.e. enumerables of type $j = 0$ following interpretation is drawn for the parameters

$$\beta_0^0 = \left(\left(K^0(0)N^0(0)\right)/\alpha^0 T_L\right) \text{ and } \beta_1^0 = \alpha^0$$

Solving Eq. (4.2.11) for n and substituting for C^0 we obtain

$$C^0 = b_0^0 \ln\left(1 + \frac{b_1^0}{b_1^j}\left(\mathrm{e}^{C^j/b_0^j} - 1\right)\right); \quad i = 1, 2, 3, 4 \tag{4.2.12}$$

Defining $a_0^0 = b_0^0$, $a_1^0 = \frac{b_1^0}{b_1^j}$, and $a_2^j = \frac{1}{b_0^j}$ (4.2.12) is rewritten as

$$C^0 = a_0^0 \ln\left(1 + a_1^j\left(\mathrm{e}^{C^j a_2^j} - 1\right)\right); \quad i = 1, 2, 3, 4 \tag{4.2.13}$$

above is a three parameter logarithmic model for defect coverage in terms of measureable test coverage metrics.

4.2.1.3 Defect Density and Reliability

Failure intensity is defined as

$$\lambda = \frac{K}{T_L} N \tag{4.2.14}$$

If N_0 be the total number of faults initially present in the program and there is no new faults introduced during testing process. Then N can be computed as

$$N = N_0(1 - C^0) \tag{4.2.15}$$

substituting from (4.2.13) we get

$$N = N_0\left(1 - a_0^0 \ln\left(1 + a_1^j\left(e^{C^j a_2^j} - 1\right)\right)\right) \tag{4.2.16}$$

hence the expected time between successive failure is given as

$$\frac{1}{\lambda} = \frac{T_L}{KN_0\left(1 - a_0^0 \ln\left(1 + a_1^j\left(e^{C^j a_2^j} - 1\right)\right)\right)} \tag{4.2.17}$$

The model can also be used for estimating reliability for the operational phase using an appropriate value of the fault exposure ratio.

4.2.2 *Enhanced NHPP Based Software Reliability Growth Model Considering Testing Coverage*

Gokhale et al. [1] gave an enhanced non-homogeneous Poisson process (ENHPP) based software reliability growth model analyzing the effect of testing coverage. An important property of their formulation is that is offers a unified scheme for the various finite failure NHPP models relying on some specific forms of coverage functions.

Notation

a'	Total number of faults present initially in the software
$c_d(t)$	Probability of detecting a fault
$c(t)$	Potential fault sites coverage rate
$\lambda(t)$	Failure intensity function
$h(t)$	Hazard function
$m(t)$	Cumulative expected number of faults detected by time t

Assumptions

1. Faults are uniformly distributed over all potential fault sites
2. When a potential fault site is sensitized at time t, any fault present at that site is detected with probability $c_d(t)$
3. Repairs are affected instantly and without introduction of new faults.

Analytically the model is described as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = a'c_d(t)\frac{\mathrm{d}}{\mathrm{d}t}c(t) \tag{4.2.18}$$

general solution of above is given as

$$m(t) = a'\int_0^t c_d(s) * c'(s)ds$$

A perfect fault detection coverage implies $c_d(t) = 1$ and perfect testing coverage implies $c(\infty) = 1$. If a probability of detecting faults is assumed to be constant (say K) and we substitute $a'k = a$ then

$$m(t) = ac(t) \tag{4.2.19}$$

Using (4.2.19) the failure intensity function is given as

$$\lambda(t) = \frac{\mathrm{d}}{\mathrm{d}t}m(t) = a\frac{\mathrm{d}}{\mathrm{d}t}c(t) \tag{4.2.20}$$

From (4.2.19) and (4.2.20) failure intensity function can be rewritten as

$$\lambda(t) = (a - m(t))\frac{c'(t)}{1 - c(t)} \tag{4.2.21}$$

This form of failure intensity implies that the failure intensity depends directly on the rate at which remaining faults are covered and inversely on the uncovered faults. The failure occurrence rate per fault or the hazard function is thus given by

$$h(t) = \frac{c'(t)}{1 - c(t)} \tag{4.2.22}$$

The time dependent form of failure occurrence rate in ENHPP incorporated the time variation in the rate at which individual fault will surface. The reliability of this model will completely be dependent on the coverage function, as the probability that no failure occurs up to time $(t + h)$ given that the last failure occurred at time t is given by

$$R(h|t) = \mathrm{e}^{-\int_t^{t+h}\lambda(s)ds} = \mathrm{e}^{-a(c(t+h)-c(t))} \tag{4.2.23}$$

Using this model Gokhale et al. [1] derived several forms of coverage functions for the various existing NHPP models. The NHPP based SRGM are characterized by their mean value functions and using (4.2.19) we can obtain the corresponding coverage function. We explain this with some examples.

Exponential Coverage Function Consider the case of GO model. The mean value function of GO model is

$$m(t) = a\left(1 - \mathrm{e}^{-bt}\right)$$

Hence the coverage and hazard function for this model is given as

$$c(t) = \left(1 - \mathrm{e}^{-bt}\right)$$

and

$$h(t) = b \tag{4.2.24}$$

Weibull Coverage Function An exponential coverage function depicts decreasing failure intensity pattern during testing, however in most practical situations failure intensity increases initially and then decreases. The generalized GO model captures this increasing–decreasing failure intensity function. The mean value function of generalized GO model is

$$m(t) = a\left(1 - \mathrm{e}^{-bt^c}\right)$$

Hence the coverage and hazard function for this model is given as

$$c(t) = \left(1 - \mathrm{e}^{-bt^c}\right)$$

and

$$h(t) = bct^{c-1} \tag{4.2.25}$$

The Weibull coverage function implies a time varying failure occurrence rate per fault. The hazard function is increasing for $c > 1$, decreasing for $c < 1$ and is constant when $\mathrm{c} = 1$.

S-shaped Coverage Function The s-shaped SRGM gives rise to s-shaped coverage functions. Consider the case of inflection s-shaped model due to Yamada et al. [10]. The mean value function for the model is

$$m(t) = a\left(1 - (1 + bt)\mathrm{e}^{-bt}\right)$$

The coverage and hazard functions for this model are

$$c(t) = \left(1 - (1 + bt)\mathrm{e}^{-bt}\right)$$

and

$$h(t) = \frac{b^2 t}{1 + bt} \tag{4.2.26}$$

Similarly various other coverage functions can be derived for the various other existing SRGM.

4.2.3 Incorporating Testing Efficiency in ENHPP

Pham and Zhang [11] extended the ENHPP, incorporating the very important concept of testing efficiency in the model. The model is a general formulation for

obtaining the coverage function for the various testing efficiency based NHPP model. The failure intensity function of the model is formulated as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \frac{c'(t)}{1-c(t)}[a(t) - m(t)] \tag{4.2.27}$$

The authors used s-shaped coverage function (4.2.26) and a linear testing time dependent fault content function to account for the fault generation i.e.

$$c(t) = \left(1 - (1+bt)\mathrm{e}^{-bt}\right)$$

and

$$a(t) = a(1+\alpha t) \tag{4.2.28}$$

Solving Eq. (4.2.27) using (4.2.28) and the initial condition $m(0) = 0$ the mean value function is obtained as

$$m(t) = a\left(1 + \alpha t - \frac{(1+bt)}{\mathrm{e}^{bt}}\right) - \frac{a\alpha(1+bt)}{b\mathrm{e}^{1+bt}}\left(\ln(1+bt) + \sum_{i=0}^{\infty}\frac{(1+bt)^{i+1}-1}{(i+1)!(i+1)}\right) \tag{4.2.29}$$

This model is in the category of pure error generation model. The mathematical form of the mean value function for the above model is complex and may find limited applications in practice for this reason. As an alternative Kapur et al. [12] suggested to use other forms of fault content function. They investigated the fault content function formulated, assuming that a constant proportion of faults removed during testing can be introduced during the debugging activities, given by $a(t) = a + \alpha m(t)$. Using this fault content function in the coverage based formulation (4.2.27), the mean value function of this SRGM is given as

$$m(t) = \frac{a}{1-\alpha}\left(1 - (1+bt)^{(1-\alpha)}\mathrm{e}^{-b(1-\alpha)t}\right) \tag{4.2.30}$$

Kapur et al. [12] also proposed a different s-shaped coverage function which converges slower than the s-shaped coverage function (4.2.26). This type of curve gives better result if the test strategy is less effective in attaining maximum coverage. This coverage function is given as

$$c(t) = \left(1 - \left(1 + bt + (bt)^2/2\right)\mathrm{e}^{-bt}\right) \tag{4.2.31}$$

The mean value function for this coverage function has been evaluated for the different forms of fault content function

Case 1 $a(t) = a$ then $m(t) = a\left(1 - \left(1 + bt + (bt)^2/2\right)\mathrm{e}^{-bt}\right)$

Case 2 $a(t) = a + \alpha m(t)$ implies

$$m(t) = \frac{a}{1-\alpha}\left(1 - \left(1 + bt + (bt)^2 \Big/ 2\right)^{(1-\alpha)} \mathrm{e}^{-b(1-\alpha)t}\right)$$

Case 3 $a(t) = a(1 + \alpha t)$ implies

$$m(t) = a(1+\alpha t) - a\left(\frac{1-(1+bt)^2}{2\mathrm{e}^{bt}}\right) - \frac{a\alpha\left(1-(1+bt)^2\right)}{b\mathrm{e}^{1+bt}}\left(\begin{array}{l} \displaystyle\sum_{n=1}^{\infty}(-1)^{n-1}\frac{\left((1+bt)^{2n-1}-1\right)}{2n-1}\sum_{j=0}^{n-1}\frac{(-1)^j}{(2j)!} \\ \displaystyle + \sum_{n=1}^{\infty}(-1)^{n-1}\frac{\left((1+bt)^{2n-1}-1\right)}{2n-1}\sum_{j=0}^{n-1}\frac{(-1)^j}{(2j+1)!} \end{array}\right)$$

Pham [13] attempted to integrate fault generation in ENHPP in a different manner, given as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \frac{c'(t)}{1-c(t)}[a - m(t)] - d(t)[a - m(t)] \tag{4.2.32}$$

where $d(t)$ denotes the fault introduction rate. The failure intensity equation (4.2.32) can be interpreted as the failure intensity, as described by ENHPP, is decreased due to fault generation proportional to the fault introduction rate in the remaining fault content.

If $d(t)$ is assumed to be a decreasing function of testing time say

$$d(t) = \frac{\mathrm{d}}{1+\mathrm{d}t} \tag{4.2.33}$$

then the mean value function of the SRGM is given as

$$m(t) = a\left(1 - \left(1 + (b+d)t + b\mathrm{d}t^2\right)\mathrm{e}^{-bt}\right) \tag{4.2.34}$$

4.2.4 Two Dimensional Software Reliability Assessment with Testing Coverage

The testing coverage functions and the SRGM described up to now have been formulated with respect to time component of the testing time. Inoue and Yamada [14] claimed that SRGM which consider software reliability growth process, depending only on the testing time may not be very useful in practice. For example

due to an ineffective and inefficient test design, the execution of certain test may not yield any reliability growth due to their failure in capturing some faults, although lots of time may be spent in the testing. One solution to this problem is to develop an SRGM which depends not only on the testing time component but also simultaneously on other reliability growth factors. Ishii and Dohi [15] proposed two dimensional software reliability growth modeling, framework by extending a modeling framework of one dimensional software reliability growth modeling, based on order statistics. In the two dimensional modeling framework the software failure occurrence times distribution is defined in two dimensions, one consisting of testing time and other being the testing effort (in terms of test execution times).

Inoue and Yamada [16] proposed a testing coverage based SRGM characterizing the relationship between the testing coverage function attainment process and the software reliability growth process. They described the time dependent behavior of test coverage considering the testing skill of the test case designers. It may be noted that the coverage attained is highly related to the skill of the test designers. In order to counter the problem of one dimensional model, Inoue and Yamada [14] extended this model in two dimensional SRGM. The approach followed for the purpose is different from Ishii and Dohi [15] as this model is based on testing coverage, a measure which takes values between 0 and 1. Following the same approach in testing coverage based model, does not conserve the theoretical means of the framework. Hence based on the notion of Cobb–Douglas production function on Weibull type SRGM a two dimensional model is developed by Inoue and Yamada [14] dividing the testing time into two factors—the testing time and testing efforts. First of all we describe the one dimensional model of Inoue and Yamada [16].

4.2.4.1 The Coverage Function

The testing coverage function is derived assuming testing coverage rate at any time t to be proportional to the difference between the attainable and the current value of the testing coverage.

$$\frac{\mathrm{d}}{\mathrm{d}t}c(t) = \beta(t)(\alpha - c(t)) \quad 0 < \alpha \leq 1, \quad \beta(t) > 0 \tag{4.2.35}$$

where α is the target value of the testing coverage to be attained. Considering an attainable value of coverage make SRGM more meaningful in practice as 100% coverage may not be feasible due to very less testability of uncovered components remaining in the later phase of testing. $\beta(t)$ is the testing coverage maturity ratio at testing time t.

The testing coverage maturity ratio is defined with respect to the test designer's skill. It is assumed that the testing skill of the test case designers increase as the ratio of the testing progress goes on. Hence the testing coverage maturity ratio at the testing time t is defined as

$$\beta(t) = b_{\mathrm{sta}}(r + (1 - r)(C(t)/\alpha)) \quad r = b_{\mathrm{ini}}/b_{\mathrm{sta}} \tag{4.2.36}$$

where b_{ini} (b_{sta}) are the initial (steady state) testing skill factor of the test case designers.

The obtained coverage function under $C(0) = 0$ is

$$c(t) = \frac{\alpha\left(1 - \mathrm{e}^{-b_{\text{sta}}t}\right)}{1 + \beta\mathrm{e}^{-b_{\text{sta}}t}}, \quad \beta = (1 - r)/r \tag{4.2.37}$$

The coverage function indicates exponential growth when $r = 1$ and S-shaped growth for $r = 0$ when the program is complex, testing skills of the testing team grows with the pace of testing and testing efforts increases more and more.

4.2.4.2 The One Dimensional SRGM

The NHPP reliability growth model formulated using the above coverage function based on the assumptions of a general NHPP model is

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) \Big/ c'(t) = b(a - m(t))$$

or

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = bc'(t)(a - m(t)) \tag{4.2.38}$$

here b is defined as the fault detection rate per attained testing coverage per fault and the corresponding mean value function is

$$m(t) = a\left(1 - \mathrm{e}^{-bc(t)}\right) \tag{4.2.39}$$

4.2.4.3 The Two Dimensional SRGM

For developing a two dimensional SRGM based on coverage function, the one dimensional time space is expanded to the two dimensions. The testing time factor of the software reliability growth is classified into the two factors

1. The factors which are related to the testing time such as the calendar time.
2. The factors which are related to the testing effort such as the testing coverage, testing domain ratio and the number of executed test cases etc.

The two dimensional SRGM based on these two factors simultaneously is formulated based on the Cobb–Douglas production (or utility) function [17, 18] applied to the testing time factor of the one dimensional conventional SRGM as follows

$$t \equiv s^v u^{1-v} \tag{4.2.40}$$

Where v represents the extent of the effect of the factor on the testing time factor of one dimensional models.

Now let $\{N(s,u);\quad s\geq 0, u\geq 0\}$ be the two dimensional stochastic process representing the number of faults detected up to the testing time s and testing time u. The two dimensional NHPP is formulated as

$$pr\{N(s,u)=n\}=\frac{(m(s,u))^{u}}{n!}\mathrm{e}^{-m(s,u)} \tag{4.2.41}$$

$$m(s,u)=\int_0^s\int_0^u \lambda(x,y)\mathrm{d}x\mathrm{d}y \tag{4.2.42}$$

where $m(s, u)$, $\lambda(x, y)$ are the mean value function and the intensity functions of the two dimensional NHPP respectively.

The mean value function of the Weibull type SRGM is given as

$$m(t)=\left(\frac{t}{\rho}\right)^{\beta}\quad 0<\beta<1,\quad \rho>0 \tag{4.2.43}$$

β, ρ are the software reliability growth shape and scale parameters and the failure intensity function for this model is given as

$$\lambda(t)=\frac{\mathrm{d}}{\mathrm{d}t}m(t)=\frac{\beta t^{\beta-1}}{\rho^{\beta}} \tag{4.2.44}$$

This is an infinite failure model as $m(\infty)=\infty$ meaning that when testing is continued for infinite time, infinite number of errors will be detected which can be accounted as imperfect debugging. Following (4.2.40) the two dimensional Weibull type SSRGM is given as

$$m(s,u)=\left(\frac{s^{v}u^{1-v}}{\rho}\right)^{\beta}\quad 0<\beta<1,\quad \rho>0 \tag{4.2.45}$$

The above model processes a very important property that if $v=1$ then (4.2.45) describes the conventional time dependent SRGM as $t=s$ while if $v=0$ it describes a testing effort dependent SRGM as here $t=u$.

4.2.5 Considering Testing Coverage in a Testing Effort Dependent SRGM

Kapur et al. [19] proposed an SRGM which defined the software reliability growth with respect to testing coverage, testing efforts and time. The failure intensity of the software during testing is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t)=\frac{\mathrm{d}m}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}t} \tag{4.2.46}$$

The model can be explained as that, the number of failures detected during testing is dependent on the testing coverage at that time. Testing coverage increases as more efforts are applied and testing efforts in turn are a function of time. Each component of the failure intensity is then defined independently.

Component 1 The rate at which additional faults are identified is directly proportional to the coverage rate of the uncovered faults and the remaining fault content. Based on this assumption the first component is defined as

$$\frac{\mathrm{d}m}{\mathrm{d}e} = b_1\left(\frac{c'}{\alpha_1 - c}\right)(a - m) \tag{4.2.47}$$

where c' is the coverage rate, α_1 is the target value of the testing coverage to be attained and c is the coverage function at time t. Coverage function of (4.2.37) is defined with respect to the test efforts as

$$c(t) = \frac{\alpha_1\left(1 - \mathrm{e}^{-b_{\mathrm{sta}}X(t)}\right)}{1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}X(t)}} \tag{4.2.48}$$

using (4.2.48) the hazard rate with coverage target α_1 is

$$\frac{c'}{\alpha_1 - c} = \frac{b_{\mathrm{sta}}}{1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}X}} \tag{4.2.49}$$

Component 2 This component relates to the number of instructions executed to the testing efforts and for the sake of simplicity it is assumed to be constant, i.e.,

$$\frac{\mathrm{d}e}{\mathrm{d}X} = b_2 \tag{4.2.50}$$

while the third component is the testing effort function defined with respect to time. Any existing test effort function (see Sect. 2.7) can be used here depending on the testing profile under consideration. Using Eqs. (4.2.47), (4.2.49) and (4.2.50), (4.2.46) can be expanded as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(X(t)) = b_1\frac{b_{\mathrm{sta}}}{1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}X(t)}}(a - m(X(t)))b_2\frac{\mathrm{d}X}{\mathrm{d}t} \tag{4.2.51}$$

The mean value function obtained under $m(0) = W(0) = 0$ is

$$m(X(t)) = a\frac{\left(1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}X(t)}\right)^{b_1b_2} - (1 + \beta)\mathrm{e}^{-b_{\mathrm{sta}}b_1b_2X(t)}}{1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}X(t)}} \tag{4.2.52}$$

This model is then extended to predict the reliability in the operational phase.

4.2.6 A Coverage-Based SRGM for Operational Phase

The model coverage and effort based model discussed in the previous section is extended by the authors for the operational phase. In order that the model can be

used for operational reliability prediction, the proportion of uncovered faults at the start of the operational phase must be considered in the failure intensity formulation and first component of the model needs to be redefined. Denoting $\widehat{m}(t)$as the number of cumulative fault detected in the operational phase the failure intensity is defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}\widehat{m}(t) = \frac{d\widehat{m}}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}t}(1 - c(T)) \tag{4.2.53}$$

where T is the release time. In the operational phase the number of fault detected/removed directly depend on the number of statements executed and not on the coverage achieved. As in this phase, fault are detected in the operational profile during execution by the users and no test case is executed with the aim of fault detection. Now for defining the first component it is assumed that along with the removal of detected faults the debuggers can also remove some additional faults. This can be modeled in the manner of Kapur and Garg [19] model for error removal formula (see Sect. 2.3). The authors of this model also proposed an alternative derivation of this phenomenon using a logistic function for the fault removal rate. For defining the first component of (4.2.53) this alternative function is used to define the fault detection rate. Along with this the possibility of imperfect debugging of type fault generation is also considered in this model.

Hence it is defined as

$$\frac{d\widehat{m}}{\mathrm{d}e} = \frac{\widehat{b}_1}{1 + \widehat{\beta}\mathrm{e}^{-\widehat{b}_1 X(t)}}(\widehat{a} + b_3 m - m) \tag{4.2.54}$$

here $\widehat{b}_1, \widehat{\beta}$ are the parameters of the logistic fault detection rate function and b_3 is the constant representing the proportion of removed faults that are generated during debugging and $\widehat{a} = (a - m(T))$, i.e. the faults remaining at the time of software release. Other components are defined as such as in the previous section. Using (4.2.54), (4.2.53) can be expanded as

$$\frac{\mathrm{d}}{\mathrm{d}t}\widehat{m}(t) = \frac{\widehat{b}_1}{1 + \widehat{\beta}\mathrm{e}^{-\widehat{b}_1 X(t)}}(\widehat{a} + b_3 m - m) b_2 \frac{\mathrm{d}X}{\mathrm{d}t}(1 - c(T)) \tag{4.2.55}$$

Mean value function of fault removal phenomenon obtained from (4.2.55) is

$$\widehat{m}(t) = \frac{\widehat{a}}{1 - b_3}\left(1 - \left(\frac{\left(1 + \widehat{\beta}\right)\mathrm{e}^{-\widehat{b}_1 X(t)}}{1 + \widehat{\beta}\mathrm{e}^{-\widehat{b}_1 X(t)}}\right)^{(1-b_3)b_2(1-c(T))}\right) \tag{4.2.56}$$

Note that $X(t)$ here represents the usage function instead of test effort function. Modeling of usage function is already explained in Sect. 2.5.

4.3 Software Reliability Growth Modeling Using the Concept of Testing Domain

Measuring software reliability with respect to the isolated testing domain is an area in software reliability modeling which has not been much explored by the researchers. The initial attempts were made by Yamada and Fujiwara [3] who proposed three types of testing domains i.e. the basic testing domain, the testing domain with skill factor and the testing domain with imperfect debugging; and then these domains were closely related to the time dependent behavior of the fault detection phenomenon of the testing.

Notation

$u(t)$	total number of faults existing in the isolated testing domain at the testing time t
$w(t)$	number of detectable faults at testing time t
v	testing domain growth rate
β	the constant fault introduction rate

4.3.1 Relating Isolated Testing Domain to Software Reliability Growth: An Initial Study

4.3.1.1 Description of Testing Domains

Basic Testing Domain

Based on the definition of testing domain, the relationship between the isolated testing domain ratio and the number of faults detected by testing is formulated according to the following assumptions.

1. The debugging process is perfect
2. The latent faults in the testing domain are distributed uniformly
3. The increasing rate of the number of detectable faults in the testing domain is proportional to the number of faults remaining in the software system outside of the isolated testing domain at arbitrary testing time

Based on these assumptions the differential equation for the basic testing domain is

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t) = v(a - u(t)) \tag{4.3.1}$$

Hence the basic testing domain under the initial condition $u(0) = 0$ is defined as

$$u(t) \equiv u_b(t) = a(1 - \mathrm{e}^{-vt}) \tag{4.3.2}$$

where $u_b(t)$ represents the basic testing domain isolated representing the exponential growth curve. The testing domain growth function $\gamma(t)$ representing the rate of the isolated testing domain growth in the software system is given as

$$\gamma(t) = \gamma_b(t) = \frac{\mathrm{d}}{\mathrm{d}t} u_b(t) = av\mathrm{e}^{-vt} \tag{4.3.3}$$

Testing Domain with Skill Factor

Testing skill represent the test case designer's potential ability (i.e. fault detection possibility), which can be measured by the total number of detectable faults. If the test case designers do not fully understand the internal structure of the software system, then the test cases designed by them will represent a narrow area of the modules and functions to be influenced and may not necessarily detect many faults. In this case, even if the test cases are executed, the testing domain does not spread at a rate which can be expected otherwise. On the other hand if a professional and experienced team having a through knowledge of the software internal structure prepares the test design and the test designed so is executed following effective strategy, the isolated testing domain can cover many modules and systems which are targeted by the test and grow speedily with an increasing rate. Hence based on the assumptions of the basic testing domain and assuming the testing domain growth rate is proportional to the number of faults existing in the testing domain, the differential equations is formulated as

$$\frac{\mathrm{d}}{\mathrm{d}t} w(t) = v(a - w(t)) \tag{4.3.4}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} u(t) = v(w(t) - u(t)) \tag{4.3.5}$$

If p is the skill factor of the test case designers, it is expected that before the start of the testing by means of test case execution some portion of the testing domain can be isolated. In such a case the initial size of the isolated testing domain cannot be zero as in case of basic testing domain. Considering $a(1 - p)$ as the size of the initial testing domain the testing domain with skill factor is obtained as

$$u(t) \equiv u_{s,p}(t) = a(1 - p(1 + vt)\mathrm{e}^{-vt}); \quad 0 \leq p \leq 1 \tag{4.3.6}$$

where $u_{s,p}(t)$ represents the isolated testing domain with skill factor spreading along an s-shaped growth curve. If size of initial testing domain is a i.e. no part of the testing domain can be isolated at the starting time of the testing phase, then the testing domain with skill factor is obtained as

$$u(t) \equiv u_s(t) = a(1 - (1 + vt)\mathrm{e}^{-vt}) \tag{4.3.7}$$

and the testing domain growth rate is given as

$$\gamma(t) = \gamma_s(t) = \frac{\mathrm{d}}{\mathrm{d}t} u_s(t) = apv^2 t\mathrm{e}^{-vt} \quad \text{or} \quad av^2 t\mathrm{e}^{-vt} \tag{4.3.8}$$

depending on the initial condition.

Testing Domain with Imperfect Debugging

Imperfect debugging is a realistic fact for almost every testing profile. The definitions of basic testing domain as well as the testing domain with skill factor, both have been formulated under the assumption of a perfect debugging environment. In practice introduction of new faults during the debugging process is frequently experienced. Yamada and Fujiwara [3] claimed that in the early stage of testing, the correction of detected faults is simple and the influenced region of the modules and functions by the fault correction is very narrow. The correction of detected faults becomes complicated with the progress of testing and the influenced region spreads widely. This phase requires more careful and skilled debugging activities for the correction of more complicated faults as compared to the earlier phase. This may lead to the increase in the introduction degree of new faults with progress of testing. However despite imperfect debugging activities, the testing domain continues to spread as the testing progresses.

Assuming an imperfect debugging condition, the testing domain is formulated as

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t) = v(\alpha(t) - u(t)) \tag{4.3.9}$$

The number of faults in the system at any moment of testing time is defined as a function of time and an exponential form of fault content is used to capture the slow introduction rate in the early phases and higher in the later stages, i.e.

$$\alpha(t) = a\mathrm{e}^{\gamma t}; \quad \gamma > 0 \tag{4.3.10}$$

From (4.3.9) and (4.3.10) the basic testing domain with imperfect debugging is defined as

$$u(t) \equiv u_i(t) = \frac{av}{\gamma + v}(\mathrm{e}^{\gamma t} - \mathrm{e}^{-vt}) \tag{4.3.11}$$

where $u_i(t)$ represents the basic testing domain with imperfect debugging, ignoring the skill of the test designers. The testing domain growth rate for this domain function is given as

$$\gamma(t) = \gamma_i(t) = \frac{\mathrm{d}}{\mathrm{d}t}u_i(t) = \frac{av}{\gamma + v}(\gamma\mathrm{e}^{\gamma t} + v\mathrm{e}^{-vt}) \tag{4.3.12}$$

4.3.1.2 Software Reliability Modeling

Yamada and Fujiwara [3] carried reliability analysis based on the three kinds of testing function domains discussed in the earlier section. The SRGM are formulated based on the following assumptions

Assumptions

1. The detected faults exist in the isolated testing domain
2. The isolated testing domain ratio increases with the progress of testing.
3. Fault detection rate is proportional to the number of faults remaining in the testing domain at testing time t

The differential equation with respect to mean value function $m(t)$ based on the above assumption and testing domain functions is

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = b(t)(u(t) - m(t)) \tag{4.3.13}$$

here $b(t)$ is defined as the time dependent fault detection rate per fault remaining in the testing domain. The mean value functions of the SRGM using the four testing domain functions (4.3.2), (4.3.6), (4.3.7) and (4.3.11) under the initial condition $m(0) = 0$ and assuming $b(t) = b$ are respectively

$$m_b(t) = a\left(1 + \frac{b\mathrm{e}^{-vt} - v\mathrm{e}^{-bt}}{v - b}\right); \quad v \neq b \tag{4.3.14}$$

$$m_{s,p}(t) = a\left(1 + \frac{bp}{v - b}\left(vt + \frac{2v - b}{v - b}\right)\mathrm{e}^{-vt} - \left(1 + \frac{bp(2v - b)}{(v - b)^2}\right)\mathrm{e}^{-bt}\right); \quad v \neq b \tag{4.3.15}$$

$$m_s(t) = a\left(\begin{array}{l} 1 + \dfrac{b}{v - b}\left(vt + \dfrac{2v - b}{v - b}\right)\mathrm{e}^{-vt} \\ -\left(\dfrac{v}{v - b}\right)^2\mathrm{e}^{-bt} \end{array}\right); \quad v \neq b \tag{4.3.16}$$

$$m_i(t) = abv\left(\begin{array}{l} \dfrac{\mathrm{e}^{\gamma t}}{(\gamma + v)(\gamma + b)} + \dfrac{\mathrm{e}^{-vt}}{(\gamma + v)(v - b)} \\ -\dfrac{\mathrm{e}^{-bt}}{(\gamma + b)(v - b)} \end{array}\right); \quad v \neq b \tag{4.3.17}$$

Kapur et al. [21] integrated learning phenomenon of testing and debugging teams in the testing domain based software reliability modeling. The general differential equation for the failure intensity is given as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = b(t)(u(t) - m(t)) \quad \text{where} \quad b(t) = \frac{b}{1 + \beta\mathrm{e}^{-bt}} \tag{4.3.18}$$

substituting the testing domain functions in (4.3.18), the mean value functions of the SRGM are given as

$$m_b(t) = \frac{a}{1 + \beta\mathrm{e}^{-bt}}\left(1 + \frac{b\mathrm{e}^{-vt} - v\mathrm{e}^{-bt}}{v - b}\right); \quad v \neq b \tag{4.3.19}$$

$$m_{s,p}(t) = \frac{a}{1+\beta e^{-bt}} \left(\begin{array}{l} 1 + \dfrac{bp}{v-b}\left(vt + \dfrac{2v-b}{v-b}\right)e^{-vt} \\ -\left(1 + \dfrac{bp(2v-b)}{(v-b)^2}\right)e^{-bt} \end{array} \right); \quad v \neq bc \tag{4.3.20}$$

$$m_s(t) = \frac{a}{1+\beta e^{-bt}} \left(\begin{array}{l} 1 + \dfrac{b}{v-b}\left(vt + \dfrac{2v-b}{v-b}\right)e^{-vt} \\ -\left(\dfrac{v}{v-b}\right)^2 e^{-bt} \end{array} \right); \quad v \neq b \tag{4.3.21}$$

$$m_i(t) = \frac{abv}{1+\beta e^{-bt}} \left(\frac{e^{\gamma t}}{(\gamma+v)(\gamma+b)} + \frac{e^{-vt}}{(\gamma+v)(v-b)} - \frac{e^{-bt}}{(\gamma+b)(v-b)} \right); \quad v \neq b \tag{4.3.22}$$

4.3.2 Application of Testing Domain Dependent SRGM in Distributed Development Environment

Testing domain dependent SRGM also finds applications in reliability estimation and prediction for software systems developed under distributed development environment. Software is realized by mapping the full set of system requirements across the various sub-systems. These subsystems are integrated to make the complete software. Independent subsystems are developed by independent teams possibly at independent locations. Such software is constituted by some pre existing (used say p in number) and otherwise new components (used say m in number).

Reliability growth of independent sub components of distributed software can be analyzed using different SRGM and the reliability growth of the full system will then be given by the joint effect of the reliability growth of the independent components. Pre existing components are usually expected to contain simple type of faults. Therefore most of the reliability studies for distributed components assume that simple exponential SRGM (GO model, [22]) can be used for analyzing their reliability growth. On the other hand the new components are assumed to contain hard and complex faults. For these kinds of faults the assumption of immediate fault removal on detection proves to be false as the time lag between their observation and isolation cannot be considered negligible. As already discussed in Sect. 2.8, the failure and removal phenomenon for these components is described by two and three stage process. Yadav et al. [23] have shown an application of testing domain dependent SRGM for estimating the reliability of new components.

Notation

a	Total fault content
a_i	Initial fault content of ith component
b_i	Proportionality constant of FDR/fault isolation rate (FIR) per fault of ith component
$b_i(t)$	Logistic learning FRR of ith component (newly developed)
$m_{ir}(t)$	Mean number of faults removed from ith component by time t
$m_{if}(t)$	Mean number of failures observed in the ith component (newly developed) by time t
$m_{iI}(t)$	Mean number of faults isolated from ith component (newly developed) by time t
η	A constant parameter in the logistic learning function
p	Number of reused components having simple type of faults
q	Number of new components having hard faults
s	Number of new components having complex faults

4.3.2.1 Model for Reused Components: the Case of Simple Faults

The fault removal process for faults lying in reused components is described by exponential SRGM, given as

$$m_{ir}(t) = a_i(1 - \mathrm{e}^{-b_i t}); \quad i = 1, 2, \ldots, p \tag{4.3.23}$$

The Goel and Okumoto [22] exponential SRGM can adequately describe the simple faults present in the reused components.

4.3.2.2 Model for New Components: the Case of Hard and Complex Faults

An SRGM for Hard Faults

The assumption of immediate removal of faults on detection seems unrealistic for the case of hard faults. The testing team may require spending more time and efforts to analyze the cause of faults in the newly developed components. The failure and removal phenomenon for hard faults is thus described by a two stage process—fault detection followed by its removal.

The failure detection process of such faults can be described by the basic testing domain function which describes the number of detectable faults in the isolated basic testing domain. On account of this using basic testing domain function (4.3.2) the failure process for hard faults in the q newly developed component is given as

$$m_{if}(t) = u_i(t) = a_i(1 - \mathrm{e}^{-v_i t}) \quad i = p + 1, \ldots, p + q \tag{4.3.24}$$

After the detection the fault is isolated and removed from its place of existence, hence the removal process of faults is described as

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{ir}(t) = b_i(t)\big(m_{if}(t) - m_{ir}(t)\big) \quad i = p+1,\ldots,p+q \tag{4.3.25}$$

using a logistic fault detection rate per remaining fault i.e. $b_i(t) = \frac{b_i}{1+\beta_i \mathrm{e}^{-b_i t}}$ the mean value function of the removal process for the components containing hard faults under the initial condition that $m_{ir}(0) = 0$ is given as

$$m_{ir}(t) = \frac{a_i}{1+\beta_i \mathrm{e}^{-b_i t}}\left(1 + \frac{b_i \mathrm{e}^{-v_i t} - v_i \mathrm{e}^{-b_i t}}{v_i - b_i}\right); \quad v_i \neq b_i, i = p+1,\ldots,p+q \tag{4.3.26}$$

The second stage of the two stage removal process describes the delayed fault removal process.

An SRGM for Complex Faults

The fault isolation and removal process of faults present in some components can be even harder than the hard faults. The faults present in such components are usually called complex faults. The time delay between the fault detection and removal is greater in the case of complex faults as compared to hard faults. On account of this, more efforts are required to isolate and then removal the complex faults. The failure and removal phenomenon for complex faults is thus described by a three-stage process—fault detection, isolation followed by removal.

The first two stages of the testing process fault detection and isolation can be described on the lines of testing domain with skill factor. Here in this case when the initial size of the isolated testing domain is considered to be zero and not $a_i(1 - p_i)$ i.e. no portion of testing domain is visible before testing is started. Based on the assumptions of the basic testing domain and assuming the testing domain growth rate is proportional to the number of faults existing in the testing domain, the differential equations are formulated as:

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{if}(t) = \frac{\mathrm{d}}{\mathrm{d}t}w_i(t) = v_i(a_i - w_i(t)) \quad i = p+q+1,\ldots,p+q+r \tag{4.3.27}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{iI}(t) = \frac{\mathrm{d}}{\mathrm{d}t}u_i(t) = v_i(w_i(t) - u_i(t)) \quad i = p+q+1,\ldots,p+q+r \tag{4.3.28}$$

Hence the mean value function of the fault isolation process is given as

$$m_{iI}(t) = a_i(1 - (1+v_i t)\mathrm{e}^{-v_i t}) \quad i = p+q+1,\ldots,p+q+r \tag{4.3.29}$$

$m_{iI}(t)$ describes the fault isolation process of the complex faults for the newly developed components.

After the detection and isolation, the fault is removed from its identified location. Hence the removal process of faults is described as:

$$\frac{\mathrm{d}}{\mathrm{d}t}m_{ir}(t) = b_i(t)(m_{iI}(t) - m_{ir}(t)) \quad i = p+q+1,\ldots,p+q+r \tag{4.3.30}$$

using same form of $b_i(t)$ the mean value function of the removal process for the components containing complex faults under the initial condition that $m_{ir}(0) = 0$ is given as

$$m_{ir}(t) = \frac{a_i}{1+\beta_i e^{-b_i t}} \left(\begin{array}{l} 1 + \dfrac{b_i}{v_i - b_i}\left(v_i t + \dfrac{2v_i - b_i}{v_i - b_i}\right)e^{-v_i t} \\ \qquad -\left(1 + \dfrac{b_i(2v_i - b_{ii})}{(v_i - b_i)^2}\right)e^{-b_i t} \end{array} \right); \quad v \neq b, \qquad (4.3.31)$$

$$i = p+q+1, \ldots, p+q+r$$

The delayed removal process described by (4.3.31) describes slower convergence as compared to the (4.3.26) which can adequately describe the case of complex faults.

4.3.2.3 Modeling Total Fault Removal Phenomenon

Total fault removal phenomenon for the software is superposition of SRGM for 'p' reused and '$q + r$' newly developed components and is given by the sum of the mean value functions of removal phenomena. From Eqs. (4.3.23), (4.3.26) and (4.3.31), the SRGM for software developed under DDE is given as

$$\begin{aligned} m(t) = & \sum_{i=1}^{p} a_i(1 - e^{-b_i t}) + \sum_{i=p+1}^{p+q} \frac{a_i}{1+\beta_i e^{-b_i t}}\left(1 + \frac{b_i e^{-v_i t} - v_i e^{-b_i t}}{v_i - b_i}\right) \\ & + \sum_{i=p+q+1}^{p+q+s} \frac{a_i}{1+\beta_i e^{-b_i t}} \left(\begin{array}{l} 1 + \dfrac{b_i}{v_i - b_i}\left(v_i t + \dfrac{2v_i - b_i}{v_i - b_i}\right)e^{-v_i t} \\ \qquad -\left(1 + \dfrac{b_i(2v_i - b_i)}{(v_i - b_i)^2}\right)e^{-b_i t} \end{array} \right) \end{aligned} \qquad (4.3.32)$$

Depending on the number of used and reused components of types containing hard and complex faults, the mean value function of the removal phenomenon for the software can be developed and applied to the distribution systems.

4.3.3 *Defining the Testing Domain Functions Considering Learning Phenomenon of Testing Team*

The testing domain functions defined in the previous section assumes a constant testing domain growth rate. However, in practice it may not be constant throughout the testing phase. Moreover, an application of testing domain function on testing data may require fitting and comparison of the different domain functions. Application of an exponential domain function may fail if the data set exhibits

s-shaped growth behaviour and vice versa. An answer to this problem is formulation of flexible SRGM which can capture both types of growth behaviour and provide a good fit to a wide range of actual data sets. Kapur et al. [24] formulated an SRGM defining a time-dependent testing-domain growth rate. Assuming a power function of testing time defines the testing growth rate, the differential equation for the flexible basic testing domain is given as:

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t) = v(t)(a - u(t)) \tag{4.3.33}$$

where $v(t) = vt^k$.

Using this time dependent testing growth rate and initial condition $u(0) = 0$, the basic testing domain function is obtained as:

$$u(t) \equiv u_b(t) = a\left(1 - \mathrm{e}^{-v\left(t^{k+1}/k+1\right)}\right) \tag{4.3.34}$$

For $k = 0, k = 1$ and $k = 2$ the testing domain function describes an exponential, Rayleigh and Weibull curve respectively. While formulating the testing domain function (4.3.2) the fault content distribution is assumed to be uniform over the software system. However it can be non-uniform. Assuming p denotes the uniformity factor in error distribution, the testing domain function can be redefined as:

$$u(t) \equiv u_{b,p}(t) = a\left(1 - p\mathrm{e}^{-v\left(t^{k+1}/k+1\right)}\right) \tag{4.3.35}$$

The testing domain growth function $\gamma(t)$ is given as:

$$\gamma(t) = \gamma_{b,p}(t) = \frac{\mathrm{d}}{\mathrm{d}t}u_b, p(t) = apvt^k\mathrm{e}^{-v\left(t^{k+1}/k+1\right)} \tag{4.3.36}$$

for the uniform testing domain substitute $p = 1$ in (4.3.36).

4.3.3.1 Testing Domain with Skill Factor

Similar to the case of flexible basic testing domain, flexible testing domain with skill factor can be formulated. Based on the assumptions of the basic testing domain, assuming the testing domain growth rate is proportional to the number of faults existing in the testing domain and is power function of testing time, the differential equations are formulated as:

$$\frac{\mathrm{d}}{\mathrm{d}t}w(t) = v(t)(a - w(t)) \tag{4.3.37}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}u(t) = v(t)(w(t) - u(t)) \tag{4.3.38}$$

where $v(t) = vt^k$.

If we consider the size of initial isolated testing domain zero, the flexible testing domain function with skill factor is given as:

$$u(t) \equiv u_s(t) = a\left(1 - \left(1 + v\left(t^{k+1}/k+1\right)\right)e^{-v\left(t^{k+1}/k+1\right)}\right) \tag{4.3.39}$$

However, if p is the skill factor of the test case designers and the initial size of the isolated testing domain is $a(1-p)$, the testing domain with skill factor is obtained as:

$$\begin{aligned} u(t) &\equiv u_{s,p}(t) \\ &= a\left(1 - p\left(1 + v\left(t^{k+1}/k+1\right)\right)e^{-v\left(t^{k+1}/k+1\right)}\right); \quad 0 \le p \le 1 \end{aligned} \tag{4.3.40}$$

Factor $p = 0$ indicates that the test designers are expert and experienced leading to ultra high potential of detecting the faults in initial stages of testing. However, the case is not true in any situation in the real testing profile.

Similarly, a flexible testing-domain function with imperfect debugging can be derived using a power function of testing time to define the testing-domain growth rate. The derivation is left as an exercise to the readers.

4.3.3.2 Software Reliability Modeling

The flexible testing domain function can be used for reliability analysis based on testing-domain growth rate. As compared to the reliability analysis carried out by Yamada and Fujiwara [3], SRGM based on flexible testing domain functions generates flexible reliability growth curves. The differential equation for the SRGM based on the assumptions of SRGM formulated in Sect. 4.3.1.2 and flexible testing domain functions is given as:

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = b(t)(u(t) - m(t)) \tag{4.3.41}$$

A power logistic form of $b(t)$ is used to defined the fault detection rate per remaining fault in order to obtain flexible SRGM considering the learning phenomenon of the testing team i.e.

$$b(t) = \frac{bt^k}{1 + \beta e^{-b\left(t^{k+1}/k+1\right)}} \tag{4.3.42}$$

Now using the four flexible testing domain functions given by (4.3.34), (4.3.35), (4.3.39) and (4.3.40), the mean value functions of the SRGM provided ($v \neq b$)are given as:

$$m_b(t) = \frac{a}{1 + \beta e^{-b\left(t^{k+1}/k+1\right)}}\left(1 + \frac{be^{-v\left(t^{k+1}/k+1\right)} - ve^{-b\left(t^{k+1}/k+1\right)}}{v - b}\right) \tag{4.3.43}$$

$$m_{b,p}(t) = \frac{a}{1+\beta e^{-b(t^{k+1}/k+1)}}\left(1+\frac{bpe^{-v\left(t^{k+1}/k+1\right)}-(v-b+bp)e^{-b\left(t^{k+1}/k+1\right)}}{v-b}\right) \tag{4.3.44}$$

$$m_s(t) = \frac{a}{1+\beta e^{-b(t^{k+1}/k+1)}}\left(\begin{array}{l}1+\frac{b}{v-b}\left(v\left(\frac{t^{k+1}}{k+1}\right)+\frac{2v-b}{v-b}\right)e^{-v\left(t^{k+1}/k+1\right)}\\ -\left(\frac{v}{(v-b)}\right)^2 e^{-b\left(t^{k+1}/k+1\right)}\end{array}\right) \tag{4.3.45}$$

$$m_{s,p}(t) = \frac{a}{1+\beta e^{-b(t^{k+1}/k+1)}}\left(\begin{array}{l}1+\frac{bp}{v-b}\left(v\left(\frac{t^{k+1}}{k+1}\right)+\frac{2v-b}{v-b}\right)e^{-v\left(t^{k+1}/k+1\right)}\\ -\left(1+\frac{bp(2v-b)}{(v-b)^2}\right)e^{-b\left(t^{k+1}/k+1\right)}\end{array}\right) \tag{4.3.46}$$

The testing-domain-dependent SRGM is not limited to those discussed here. Several SRGMs can be formulated based on the discussed testing domain functions integrating the various other concepts of software reliability modeling and varying the fault detection rate function.

4.4 Data Analysis and Parameter Estimation

Both testing coverage and testing domain based models find application for a specific purpose. Coverage-based models are useful when one wants to know the progress of the testing coverage attained, while testing domain based models provide information related to the isolated testing domain and the reliability measures. In this section we have established the validity of several models from both category and estimated their parameters and drawn comparisons on the estimation results. Due to the absence of coverage and testing domain growth rate related actual life observations in the collected data sets, directly software reliability models are fitted on the data sets and using the estimates of SRGM, an estimate of the attained coverage or the testing domain isolated can be made.

4.4.1 Application of Coverage Models

Failure Data Set

The failure data is for the program that monitors a real-time control system consists of about 200 modules having on average, 1,000 lines of a high level language

such as Fortran. The test data for 111 days is available during which 481 faults were detected [25]. Since the test data is recorded daily, the test operations performed in a day are regarded to be a test instance.

Following models have been chosen for data analysis and parameter estimation.

Model 1 (M1) Exponential Coverage SRGM [1]

$$c(t) = \left(1 - \mathrm{e}^{-bt}\right); \quad m(t) = a\left(1 - \mathrm{e}^{-bt}\right)$$

Model 2 (M2) Weibull Coverage SRGM [1]

$$c(t) = \left(1 - \mathrm{e}^{-bt^c}\right); \quad m(t) = a\left(1 - \mathrm{e}^{-bt^c}\right)$$

Model 3 (M3) S-shaped Coverage SRGM [1]

$$c(t) = \left(1 - (1 + bt)\mathrm{e}^{-bt}\right); \quad m(t) = a\left(1 - (1 + bt)\mathrm{e}^{-bt}\right)$$

Model 4 (M4) Testing Efficiency Coverage SRGM [11]

$$c(t) = \left(1 - (1 + bt)\mathrm{e}^{-bt}\right); \quad m(t) = \frac{a}{1 - \alpha}\left(1 - (1 + bt)^{(1-\alpha)}\mathrm{e}^{-b(1-\alpha)t}\right)$$

Model 5 (M5) S-shaped Coverage SRGM [12]

$$c(t) = \left(1 - \left(1 + bt + (bt)^2\Big/2\right)\mathrm{e}^{-bt}\right);$$
$$m(t) = a\left(1 - \left(1 + bt + (bt)^2\Big/2\right)\mathrm{e}^{-bt}\right)$$

Model 6 (M6) Testing Efficiency S-shaped Coverage SRGM [12]

$$c(t) = \left(1 - \left(1 + bt + (bt)^2\Big/2\right)\mathrm{e}^{-bt}\right);$$
$$m(t) = \frac{a}{1 - \alpha}\left(1 - (1 + bt + (bt)^2\Big/2)^{(1-\alpha)}\mathrm{e}^{-b(1-\alpha)t}\right)$$

Model 7 (M7) Testing Efficiency S-shaped Coverage SRGM [13]

$$c(t) = \left(1 - (1 + bt)\mathrm{e}^{-bt}\right); \quad m(t) = a\left(1 - \left(1 + (b + d)t + bdt^2\right)\mathrm{e}^{-bt}\right)$$

Model 8 (M8) Flexible Coverage Function SRGM [16]

$$c(t) = \frac{\alpha\left(1 - \mathrm{e}^{-b_{\mathrm{sta}}t}\right)}{1 + \beta\mathrm{e}^{-b_{\mathrm{sta}}t}}, \quad \beta = (1 - r)/r;$$
$$m(t) = a\left(1 - \mathrm{e}^{-bc(t)}\right)$$

The unknown parameters of all these models have been estimated using the regression module of SPSS. The values of estimated parameters have been tabulated in Table 4.1. Figures 4.2 and 4.3 show the goodness of fit curves for the estimation results tabulated in Table 4.1 and future predictions for SRGM M1–M4 and M5–M8 respectively.

The estimation results shows that exponential SRGM does not give a good fit on the data. The mean square value of the estimated values is very high for the

Table 4.1 Estimation result for model 1 to model 8

Model	Estimated parameters					Comparison criteria	
	a	b	c, d, $b_{s\ t\ a}$	α	β	MSE	R^2
M1	542	0.0254	1.4993	–	–	804.93	0.965
M2	484	0.0054	–	–	–	300.85	0.987
M3	489	0.0662	–	–	–	331.76	0.985
M4	458	0.0690	–	0.0640	–	334.65	0.985
M5	476	0.1045	–	–	–	555.25	0.975
M6	450	0.1080	0.0000	0.0540	–	548.60	0.976
M7	489	0.0660	0.0649	–	–	335.06	0.985
M8	815	0.9515	1.4993	0.9515	5.85	307.58	0.965

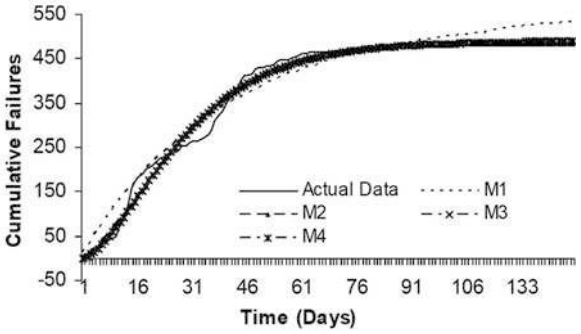


Fig. 4.2 Goodness of fit curve for model M1–M4

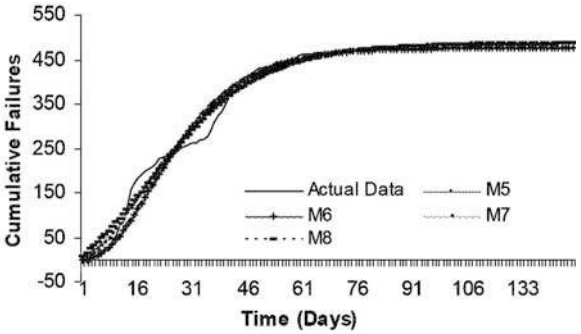


Fig. 4.3 Goodness of fit curve for model M5–M8

exponential SRGM M1. However the model M8 is also exponential but due to the flexible coverage function the model has given a good fit. The s-shaped coverage function of models M5 and M6 $c(t) = (1 - (1 + bt + (bt)^2/2)\mathrm{e}^{-bt})$ also does not give a good fit. This coverage function converges slower than the s-shaped coverage function $c(t) = (1 - (1 + bt)\mathrm{e}^{-bt})$. The results of the models M3, M4 and M7 are comparable. Both Weibull-type coverage function and flexible-coverage-function based SRGM have given a good fit on the data. The shape parameter in both of these coverage functions offers flexibility to capture a wide range of coverage curves. The estimates of model M2–M4 and M7 and M8 are stable and

Table 4.2 Coverage results for model 1 to model 8

Model	Software coverage attained in 109 days	m (111)
M1	0.9375	509
M2	0.9978	484
M3	0.9939	486
M4	0.9954	486
M5	0.9999	475
M6	0.9999	475
M7	0.9938	486
M8	0.9461	484

consistent with the collected data as the estimated value of the number of failure converges between 483 and 485 faults for these models and not much further increase in the failure number is observed for these models. This data analysis concludes that model M2 should be selected for future prediction and reliability measurement.

These models also give us the information on achieved level of software coverage with the progress of testing. All of the s-shaped models show that up to the time period for which the data was observed i.e. 120 days more than 99% of software coverage is attained. While the exponential models shows less coverage as compared to the s-shaped models (for model M1 93.75% and M8 94.61% in 109 days). If the developer decide to terminate the testing on the bases of the coverage information then according to the best fit model 99.78% has been attained which means that after 109 days of testing the software can be released. The summary of coverage information is tabulated in Table 4.2.

4.4.2 Application of Testing Domain Based Models

Failure Data Set

We again continue our data analysis on the data set used in Sect. 3.8.1 in this section. This data set is an interval domain data cited in Brooks and Motley [26]. The failure data set is for a radar system of size 124 KLOC (Kilo Lines of Code) tested for 35 weeks in which 1,301 faults were detected.

Following models have been chosen for data analysis and parameter estimation.

Yamada and Fujiwara [3] Testing Domain based Models

Model 9 (M9) $m_b(t) = a\left(1 + \frac{be^{-vt} - ve^{-bt}}{v-b}\right); \quad v \neq b$

Model 10 (M10)

$$m_{s,p}(t) = a\left(1 + \frac{bp}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt} - \left(1 + \frac{bp(2v-b)}{(v-b)^2}\right)e^{-bt}\right); \quad v \neq b$$

Model 11 (M11) $m_s(t) = a\left(1 + \frac{b}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt} - \left(\frac{v}{v-b}\right)^2 e^{-bt}\right); \quad v \neq b$

Model 12 (M12) $m_i(t) = abv\left(\frac{e^{\gamma t}}{(\gamma+v)(\gamma+b)} + \frac{e^{-vt}}{(\gamma+v)(v-b)} + \frac{e^{-bt}}{(\gamma+b)(v-b)}\right); \quad v \neq b$

Kapur et al. [21] Testing Domain Based SRGM Incorporating Learning Phenomenon

Model 13 (M13) $m_b(t) = \frac{a}{1+\beta e^{-bt}}\left(1 + \frac{be^{-vt} - ve^{-bt}}{v-b}\right); \quad v \neq b$

Model 14 (M14) $m_{s,p}(t) = \frac{a}{1+\beta e^{-bt}}\left(1 + \frac{bp}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt} - \left(1 + \frac{bp(2v-b)}{(v-b)^2}\right)e^{-bt}\right); \quad v \neq b$

Model 15 (M15) $m_s(t) = \frac{a}{1+\beta e^{-bt}}\left(1 + \frac{b}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt} - \left(\frac{v}{v-b}\right)^2 e^{-bt}\right); \quad v \neq b$

Model 16 (M16)

$$m_i(t) = \frac{abv}{1+\beta e^{-bt}}\left(\frac{e^{\gamma t}}{(\gamma+v)(\gamma+b)} + \frac{e^{-vt}}{(\gamma+v)(v-b)} + \frac{e^{-bt}}{(\gamma+b)(v-b)}\right); \quad v \neq b$$

Kapur et al. [24] Learning Testing Domain Based SRGM

Model 17 (M17) $m_b(t) = \frac{a}{1+\beta e^{-b\left(t^{k+1}/k+1\right)}}\left(1 + \frac{be^{-v\left(t^{k+1}/k+1\right)} - ve^{-b\left(t^{k+1}/k+1\right)}}{v-b}\right)$

Model 18 (M18)

$$m_{b,p}(t) = \frac{a}{1+\beta e^{-b(t^{k+1}/k+1)}}\left(1 + \frac{bpe^{-v\left(t^{k+1}/k+1\right)} - (v-b+bp)e^{-b\left(t^{k+1}/k+1\right)}}{v-b}\right)$$

Model 19 (M19)

$$m_s(t) = \frac{a}{1+\beta e^{-b\left(t^{k+1}/k+1\right)}}\left(1 + \frac{b}{v-b}\left(v\left(\frac{t^{k+1}}{k+1}\right) + \frac{2v-b}{v-b}\right)e^{-v\left(t^{k+1}/k+1\right)} - \left(\frac{v}{(v-b)}\right)^2 e^{-b\left(t^{k+1}/k+1\right)}\right)$$

Table 4.3 Estimation result for model 9 to model 20

Model	Estimated parameters						Comparison criteria	
	A	b	v	p, γ	β	k	MSE	R^2
M9	1,689	0.0899	0.0899	–	–	–	2967.63	0.955
M10	1,457	0.1656	0.1630	0.9866	–	–	1370.24	0.987
M11	1,449	0.1662	0.1663	–	–	–	1343.33	0.985
M12	1,865	0.9700	0.0320	0.0042	–	–	20611.25	0.945
M13	1,425	0.1715	0.1096	–	4.0055	–	973.89	0.996
M14	1,342	0.1938	0.2214	0.3045	21.2299	–	5747.39	0.976
M15	1,336	0.1977	0.1756	–	0.6292	–	2413.85	0.998
M16	1,720	0.6550	0.0740	0.0280	97.6590	–	9979.99	0.967
M17	1,322	0.1204	0.3280	–	10.7697	0.1752	1289.15	0.994
M18	1,318	0.0054	0.3215	0.2405	0.2926	1.1234	657.71	0.999
M19	1,401	0.2304	0.1164	–	0.2671	0.1077	1341.69	0.989
M20	1,426	0.2304	0.1033	0.7934	8.2962	0.1000	1136.28	0.996

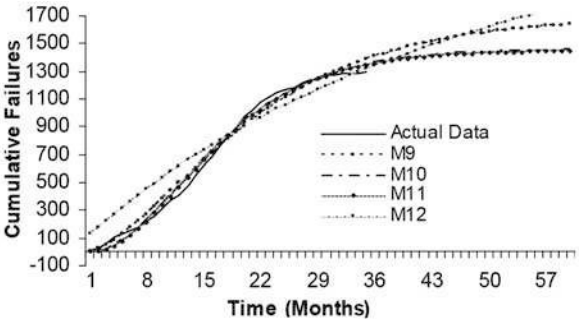


Fig. 4.4 Goodness of fit curve for models M9–M12

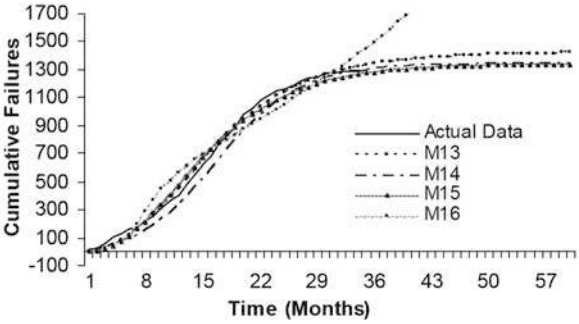


Fig. 4.5 Goodness of fit curve for models M13–M16

Model 20 (M20)

$$m_{s,p}(t) = \frac{a}{1+\beta e^{-b\left(t^{k+1}/k+1\right)}}\left(1+\frac{bp}{v-b}\left(v\left(\frac{t^{k+1}}{k+1}\right)+\frac{2v-b}{v-b}\right)e^{-v\left(t^{k+1}/k+1\right)} - \left(1+\frac{bp(2v-b)}{(v-b)^2}\right)e^{-b\left(t^{k+1}/k+1\right)}\right)$$

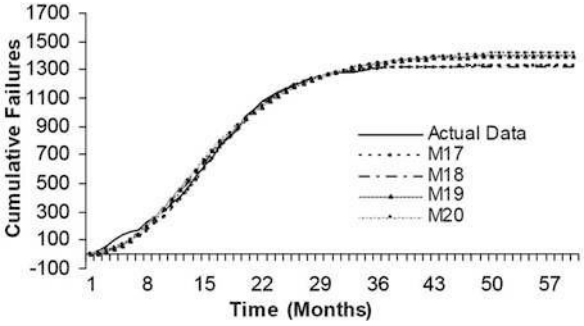


Fig. 4.6 Goodness of fit curve for models M17–M20

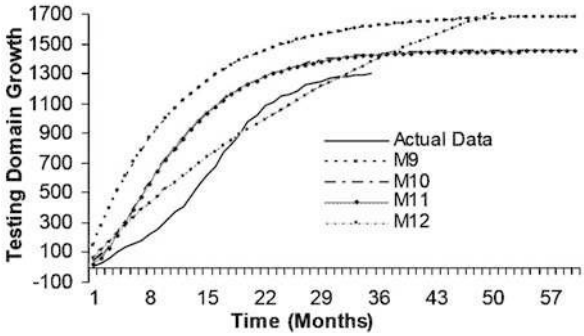


Fig. 4.7 Testing domain growth curve for models M9–M12

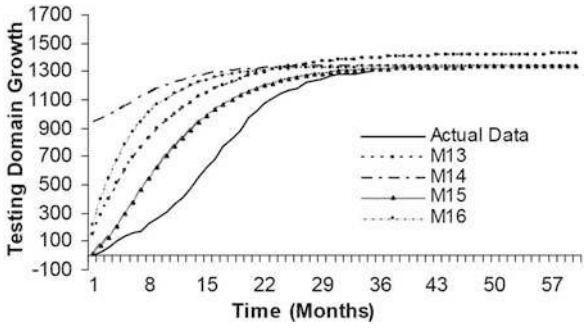


Fig. 4.8 Testing domain growth curve for models M13–M16

The unknown parameters of these models have been estimated and are tabulated in Table 4.3. Figures 4.4, 4.5, and 4.6 show the goodness of fit curves for the estimation results tabulated in Table 4.3 and future predictions and Figs. 4.7, 4.8, and 4.9 shows the growth of testing domain with the progress of testing for the three types of the testing domain models.

The data analysis results on testing domain dependent models show lots of variability. Mean square error for models M9, M12, M14, M15 and M16 is very high. Hence none of these models can be chosen for reliability measurements. Both of the imperfect debugging models fitted very poorly on this data set. As the

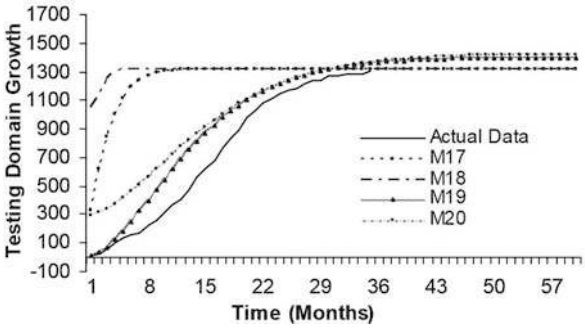


Fig. 4.9 Testing domain growth curve for models M17–M20

observed data shows an s-shaped growth pattern poor fit is observed mostly for exponential models. The model that fits best on this data is M18 and none of other model has comparable value of MSE or R^2. The best fit model is based on the testing domain function that considers the learning of testing team and basic testing domain with non zero size of initial testing domain. For the best fitted model the size of faults in the software is estimated to be 1,318. The fault detection rate is 0.0055 and the testing domain growth rate is 0.3215. The learning factor is estimated to be 1.1234 in the testing domain function and 0.2926 in the reliability growth model. The result implies that learning occurs both during testing domain isolation as well as fault debugging.

Exercises

1. What is testing coverage? Give some important measures of testing coverage.
2. Explain the difference between path and branch coverage.
3. What is testing domain? How efficiency of test case design is related to the isolated testing domain?
4. Show that if the failure intensity function of an SRGM is given as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \frac{c'(t)}{1-c(t)}[a(t) - m(t)],$$

   where $c(t) = (1 - (1 + bt)\mathrm{e}^{-bt})$ and $a(t) = a + \alpha m(t)$, then the mean value function of the SRGM is given as $m(t) = \frac{a}{1-\alpha}\left(1 - (1+bt)^{(1-\alpha)}\mathrm{e}^{-b(1-\alpha)t}\right)$.
5. Show with the help of graph, similarity and difference in the two coverage functions $c(t) = (1 - (1 + bt)\mathrm{e}^{-bt})$ and $c(t) = (1 - (1 + bt + (bt)^2/2)\mathrm{e}^{-bt})$.
6. Model discussed in Sect. 4.2.5 defines an SRGM with respect to coverage, testing effort and time. The failure intensity of the software during testing is defined as $\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \frac{\mathrm{d}m}{\mathrm{d}e}\frac{\mathrm{d}e}{\mathrm{d}X}\frac{\mathrm{d}X}{\mathrm{d}t}$. The second component of the model relates the number of instructions executed to the testing effort, i.e. it can be called as the statement coverage with respect to the testing efforts. Develop a statement coverage model, if we define $\frac{\mathrm{d}m}{\mathrm{d}e} = (b_1 + b_2(m/a))(a - m)$ and $\frac{\mathrm{d}e}{\mathrm{d}X} = b_3X^k$, with the initial condition $m(0) = W(0) = 0$.

Week	Old component	New component 1	New component 2
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

7. Assume that software is developed in distributed development environment with one old and two new components. The old component contains simple faults. Fault content of one new component is of hard type while in the other it is of complex type. Fit the testing domain dependent SRGM for distributed development environment discussed in the Sect. 4.3.2. Base your analysis on the following failure data.
   Calculate the mean square error and variation in the estimated parameters.

References

1. Gokhale SS, Philip T, Marinos PN, Trivedi KS (1996) Unification of finite failure non-homogeneous poisson process models through test coverage. In: Proceedings 7th International Symposium on Software Reliability Engineering, White Plains, pp 299–307
2. Malaiya YK, Li MN, Bieman JM, Karcich R (2002) Software reliability growth with test coverage. IEEE Trans Reliab 51(4):420–426
3. Yamada S, Fujiwara T (2001) Testing-domain dependent software reliability growth models and their comparisons of goodness-of-fit. Int J Reliab Qual Saf Eng 8:205–218
4. Fujiwara T, Yamada S (2001) Software reliability growth modeling based on testing skill characteristics: Model and Application. Electron Commun Jpn 84(3):42–48
5. Malaiya YK, Yang S (1984) The coverage problem for random testing. Proc Int Test Conf ITC-84:237–245
6. Ramsey J, Basili (1985) Analyzing the test process using structural coverage. In: Proceedings 8th International Conference on Software Engineering, London, England, 306–312
7. Malaiya YK, Karunanithi N, Verma P (1992) Predictability of software reliability models. IEEE Trans Reliab 41:539–546
8. Malaiya YK, Mayrhauser A, Srimani P (1993) An examination of fault exposure ratio. IEEE Trans Softw Eng 19(11):1087–1094
9. Malaiya YK, Li N, Bieman J, Karcich R, Skibbe B (1994) The relationship between test coverage and reliability. In: Proceedings of the 5th International Symposium Software Reliability Engineering, Monterey, CA, pp 186–195
10. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R-32(5):475–484

11. Pham H, Zhang X (2003) NHPP software reliability and cost models with testing coverage. Eur J Oper Res 145(2):443–454
12. Kapur PK, Singh O, Gupta A (2005) Some modeling peculiarities in software reliability. In: Proceedings Kapur PK, Verma AK (eds) Quality, reliability and infocom technology, trends and future directions. Narosa Publications Pvt. Ltd., New Delhi, pp 20–34
13. Pham H (2006) System software reliability., Reliability engineering seriesSpringer Verlag, London
14. Inoue S, Yamada S (2008) Two dimensional software reliability assessment with testing coverage. The 2nd International Conference on Secure System Integration and Reliability Improvement, pp 150–155
15. Ishii T, Dohi T, (2006) Two-Dimensional Software Reliability Models and Their Application. In: Proceedings 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06), pp 3–10
16. Inoue S, Yamada S (2004) Testing coverage dependent software reliability growth modeling. Int J Reliab Qual Saf Eng 11(4):303–312
17. Varian HR (1991) Intermediate microeconomics—a modern approach, 2nd edn. WW Norton & Company, New York
18. Ahn CW, Chae KC, Clark GM (1998) Estimating parameters of the power law process with two measures of failure time. J Qual Technol 30(2):127–132
19. Kapur PK, Singh O, Bardhan A (2005) A software reliability growth model for operational use with testing coverage. In: Kapur PK, Verma AK (eds) Quality, reliability and IT (trends and future directions). Narosa Publications Pvt. Ltd., New Delhi, pp 60–73
20. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
21. Kapur PK, Khatri S, Gupta A, Singh VB (2006) Flexible testing-domain dependent software reliability growth models. In Proceedings Varde PV, Srividya A, Sanyasi Rao VVS, Chauhan A (eds) Reliability safety and hazards: advances in risk-informed technology, Narosa Publications Pt. Ltd., New Delhi, pp 166–174
22. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R-28(3):206–211
23. Yadav K, Goswami DN, Kapur PK (2007) Testing-domain based software reliability growth models for distributed environment. In: Proceedings 3rd International Conference on Reliability and Safety Engineering (INCRESE-2007), Udaipur, pp 614–628
24. Kapur PK, Yadav K, Singh O, Yadavalli VSS (2007) Testing-domain dependent software reliability growth models with power-logistic function. In: Kapur PK, Verma AK (eds) Quality reliability and infocom technology. MacMillan India Ltd., pp 284–294
25. 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 Softw Eng SE-17:483–489
26. Brooks WD, Motley RW (1980) Analysis of discrete software reliability models—technical report (RADC-TR-80-84). Rome Air Development Center, New York

Chapter 5
Change-Point Models

5.1 Introduction

Detection of a failure and successful removal of the fault that has caused the failure during software testing are affected by many factors. These factors include testing environment, strategy, testing team constitution and efficiency, test case effectiveness, resources and many more. The software reliability models formulated to track the reliability growth during testing consider a few or a number of these factors. Drawing certain assumptions on the testing process the models are formulated. The model parameters are representative of the various factors about the reliability growth and depict specified factors about the phenomenon under consideration, that is software testing. In the applications of SRGM on real testing environment to estimate the reliability for the period of testing it is assumed that the parameters of the SRGM can remain smooth over the testing period. However it may not be the case. For example, consider the situation that after two days of testing and analyzing the failure data the developer management may decide that an additional highly professional member should join the testing team and they also change the existing testing strategy and use some new automated testing tool. All these efforts are done to pace the testing progress. At this point of time if the model parameters estimated before changes are implemented may not describe the further testing progress adequately. Magnitude of some model parameters may change. Such changes are often observed in the testing environments.

From the point of view of statisticians, the typical situation of change as defined by change-point models defines a particulars phenomenon in question according to some structural, physical and environmental factors, for different time periods or spatial regions different set of model parameter values may be needed in order to describe the reality adequately. The time points which separate the time periods are called *change points*. In the regression literature, the change-point model is also referred to as two- or multiple-phase regression, switching regression, segmented regression, two-stage least squares or broken-line regression. Important

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_5,
© Springer-Verlag London Limited 2011

contexts in question here are to find—Is it necessary to assume that the parameters are changing? When or where does a change occur? Does the change take place over a certain period of time? What is the onset and duration of change? Do some or all of the parameters of the model changes? How much do parameters before and after the change point differ? What type of model is appropriate in a particular situation? What are the models in the literature that can be applied for these situations?—etc. [1, 2].

The bibliography evidence suggests mainly two types of techniques to handle these variations. First, diagnostic checking methods to identify non-consistencies using regression or time series analysis methods. Second, incorporating parametric variability in models. Variability of parameters in models being the essential consideration. The change-point problem was first introduced in the quality control context, which was concerned about the output of a production line and wanted to find any departure from an acceptable standard of the products. Although traditionally, control charts are used to detect changes. The major difference between a change-point analysis and a control chart in this concern is that the control chart is intended to be updated following the collection of each data point. A change-point analysis is intended to be performed less frequently to review the performance over a more extended period of time. The two methods can be used in a complementary fashion. Change-point analysis determined the number of changes and estimates the time of each change. It further provides confidence levels for each change and confidence intervals for the time of each change. Since then the change-point problem has developed into a fundamental problem in the areas of statistical control theory, stationarity of a stochastic process, estimation of the current position of a time series, testing and estimation of change in the patterns of a regression model, and most recently in the comparison and matching of DNA sequences in micro array data analysis [2]. The problem of abrupt/smooth changes is often encountered in various experimental and mathematical sciences.

Change-point models are also very important for the hardware and software reliability study. In the vast literature of software reliability modeling, most researchers assume a constant detection rate per fault in deriving their models. It is supposed that all faults have equal probability of being detected during the software testing process, and the rate remains constant over the intervals between fault occurrences. In reality, the fault detection rate strongly depends on the skill of test teams, program size, defect density, code expansion factor, testing efforts in terms of CPU hours and team constitution and software testability. Therefore, it may not be smooth and can be changed. On the other hand, if we want to detect more faults in the software in order to reach the desired reliability objective during testing and meet the scheduled deliveries it is advisable to purchase new equipments or introduce new tools/techniques, which are fundamentally different from the methods currently in use, if the software companies can afford a larger budget for testing and debugging. These external new methods can give a detailed description of the test methodology,

a complete test report, or an expert analysis of the findings to the clients. These approaches can provide a conspicuous improvement in software testing and productivity. In this case, the fault detection rate will be changed during the software development process whenever any change is brought in the testing process. In addition to all these factors another reason for observing changes is the change of the software life cycle phase. We know that software reliability continues to grow even during the operational phase due to the field failures. A change-point may occur when the software life cycle phase changes from the testing phase to the operational phase. It is more appropriate to use the change-point method for reliability analysis in the changing testing process. SRGM, which do not consider the effect of change-point in software reliability estimation may not express the factual software reliability behavior [3, 4].

First we describe mathematically the theory of change-point analysis [3, 5] and then discuss the study carried in software reliability modeling on the lines of change-point analysis.

A change-point exists in the observations from a sequence of random variables $X_1, X_2, \ldots, X_n$ when the observations $X_1, X_2, \ldots, X_\tau$ follow the same distributions F, while the observations $X_{\tau+1}, X_{\tau+2}, \ldots, X_n$ follow the distribution G, such that $F \neq G$. The index τ is called the change-point. This describes a particular situation which implies the existence of a single change in the phenomenon under observation. In general the multiple change-points are said to exist in the observations from a sequence of random variables $X_1, X_2, \ldots, X_n$ when the observations $X_1, X_2, \ldots, X_{k1}$ follow the distribution F_1, observation $X_{k1+1}, X_{k1+2}, \ldots, X_{k2}$ follows the distribution F_2, such that $X_{k1} \neq X_{k1+1}$, and so on that observation $X_{kq+1}, X_{kq+2}, \ldots, X_n$ follows the distribution F_q, such that $X_{kq} \neq X_{kq+1}$. Here $1 < k_1 < k_2 < \cdots < k_q < n$, q is the number of change-points and $k_1, k_2, \ldots, k_q$ are respectively the positions of the change-point. The distributions $F_1, F_2, \ldots, F_q$ may or may not belong to the same parametric family $F(\theta)$, $\theta \in R^p$. In case they belong to the same parametric family then the change-point problem is defined as that the observations $X_1, X_2, \ldots, X_{k1}$ follows the distribution F, with parameter θ_1, observation $X_{k1+1}, X_{k1+2}, \ldots, X_{k2}$ also follows the distribution F but the parameters of the distribution have changed, say represented as θ_2, and so on that observation $X_{kq+1}, X_{kq+2}, \ldots, X_n$ follows the distribution F with parameters θ_q. Again $1 < k_1 < k_2 < \cdots < k_q < n$, q is the number of change-points and $k_1, k_2, \ldots, k_q$ are, respectively, the positions of the change-point.

The studies related to the change-point analysis were mostly in general related to estimating the position of change-point, determining the number of change-points and their respective position in case of multiple change-points and determining the distributions or the parameters in case the distribution of observations between two change-points is same for each set of observations. In the statistical literature the change-point problem has widely been studied by many authors. Hinkley [6] used maximum likelihood estimation to estimate the single change-point for the case the distributions F and G are from the same parametric family or may be arbitrary known distributions. Carlstein [7] discussed the non-parametric

estimation of the change-point. Joseph and Wolfson [8] discussed the generalized concept by studying the multi-path change-points, where several independent sequences are considered simultaneously with one change-point. Chen and Gupta [5] carried out test of significance to establish whether a change-point exists or not and also estimated it if it existed. Xie et al. [9] discussed change points of mean residual life and hazard functions for certain generic Weibull distributions. Bae and Kvam [10] showed that reliability estimation could be improved substantially by using the change-point model to account for product burn-in effects. Zhao and Wang [11] described new test statistics to test the existence of change-points in software reliability models. Galeano [12] proposed the use of cumulative sums for detections of change-points of a Poisson process when the failure rate is piecewise constant.

Change-point models are very useful for hardware and software reliability studies. Initial studies in change-point based reliability modeling were carried out jointly for hardware and software reliability analysis due to Zhao [3]. Later many researchers formulated various change-point SRGM for the software reliability measurement and prediction [13–24]. In the software reliability literature the change-point analysis is mainly related to the parameter variation modeling for different time intervals between change-points. The number and time horizon of change-point is mainly determined by observation method from the failure data sets, while some studies advocate treating them as unknown parameters of the model and estimating them together with the other parameters of the model. Software package change-point analyzer can also be used for the purpose of change-point estimation. Most of the researchers in the SRGM advocate to treat them as known values in the model since the SRGM are applied to the observed failure data sets of the real life projects and the reasons of change, number of changes and their time horizon can be found from the failure data plots and obtained from the testing and debugging teams. The content of this chapter is addressed to the description of change-point models in software reliability modeling literature.

Notation

a, a_i	Expected initial software fault content (i denoting module/fault type), a, $a_i > 0$
$b(t)$, $b_i(t)$	Time-dependent rate of fault removal per remaining faults
τ, τ_i	Change-point(s)
$F(t)$, $F_i(t)$	Probability distribution function of failure times, with density function $F(t) = \int_0^t f(t)\,\mathrm{d}t$
N_0	Finite, initial fault content in the software
$m(t)$	Expected number of faults detected in the time interval $(0, t]$
$\lambda(t)$	Failure intensity function
$R(x\|t)$	Reliability function for time x given t

5.2 Change-Point Models: An Initial Study

This study [3] is related to single change-point modeling for software reliability estimation using the parametric variability approach. Let F_1 and F_2 be two different lifetime distributions with density functions $f_1(t)$ and $f_2(t)$, $X_1, X_2, \ldots, X_n$ be the inter-failure times of the sequential failures in a lifetime testing. A change-point exists in the observations from a sequence of random variables $X_1, X_2, \ldots, X_\tau, X_{\tau+1}, X_{\tau+2}, \ldots, X_n$. Three models are proposed in the study assuming the failure time distribution to be exponential, Weibull and Pareto. The exponential and Pareto distribution models are those proposed due to Jelinski and Moranda (known as JM model) [25] and Littlewood [26]. Before we describe the change-point models first we briefly describe the JM model.

This model belongs to a class of exponential order statistic model that assumes that fault detection and correction begins when a program contains a number of faults and all the faults have the same rate of detection Φ. The basic assumptions of the model are

1. The rate of fault detection is proportional to the current fault content of the software.
2. The fault detection rate remains constant over the intervals between fault occurrences.
3. A fault is corrected instantaneously without introducing new faults.
4. The software is operated in a similar manner as that in which reliability predictions are to be made.
5. Every fault has the same chance of being encountered within a severity class as any other fault in that class.
6. The failures, when the faults are detected, are independent.

If the time between failure occurrences is $x_i = t_i - t_{i-1}$, $i = l, \ldots, n$, then x_i's are independent exponentially distributed random variables with mean. Let $f(t_i)$ be probability density function for particular time t_i such that

$$f(x_i/t_{i-1}) = \Phi(a-(i-1))\mathrm{e}^{-\Phi(a-(i-1))x_i} \tag{5.2.1}$$

and cumulative density function be

$$F(t_i) = 1 - \mathrm{e}^{-\lambda_i t_i}; \quad 1/\Phi(a-(i-1)) = 1/\lambda_i \tag{5.2.2}$$

where λ_i is the hazard rate. Equations (5.2.1) and (5.2.2) imply

$$m(t) = a\left(1 - \mathrm{e}^{-\Phi t}\right) \tag{5.2.3}$$

$$\lambda(t) = a\Phi \exp^{-\Phi t} \tag{5.2.4}$$

where $m(t)$ is the mean value function and $\lambda(t)$ is the failure density function.

Now the change-point model for the JM models is formulated under the following assumptions along with the other assumption of the model.

1. There are a finite number a of items under the testing (faults), may or may not be known.
2. At the beginning all of the items have the same lifetime distribution F. After τ failures are observed the remaining $(a - \tau)$ items have the distribution G. The change-point τ is assumed unknown.
3. The sequences $X_1, X_2, \ldots, X_\tau$ and $X_{\tau+1}, X_{\tau+2}, \ldots, X_n$ are statistically independent.

Note that JM model is a finite failure model as $\lim_{t\to\infty} m(t) = a$.

5.2.1 Change-Point JM Model

If F_1 and F_2 are exponentially distributed with parameters λ_1 and λ_2, respectively, then the inter-failure times $X_1, X_2, \ldots, X_n$ are independently exponentially distributed. Specifically, X_i is exponentially distributed with parameter $\lambda_1(a - (i - 1))$, $i = 1,2,\ldots,\tau$, and X_j is exponentially distributed with parameter $\lambda_2(a - (\tau + j - 1))$, $j = \tau + 1, \tau + 2, \ldots, n$. When the observations $X_1, X_2, \ldots, X_\tau$ follow the same distributions F, the observations $X_{\tau+1}, X_{\tau+2}, \ldots, X_n$ follow the distribution F_2, such that $F_1 \neq F_2$. The index τ is called the change-point.

5.2.2 Change-Point Weibull Model

Assume F_1 and F_2 are Weibull distribution functions with parameters (λ_1, β_1) and (λ_2, β_2), respectively. That is

$$F_1(t) = \left(1 - e^{-\lambda_1 t^{\beta_1}}\right) \tag{5.2.5}$$

$$F_2(t) = \left(1 - e^{-\lambda_2 t^{\beta_2}}\right) \tag{5.2.6}$$

In this case, the time intervals of failures are dependent. The Weibull model without change-points is used by Wagoner [27] to describe the fault detection process. Particularly, when the shape parameter $\beta = 2$, the Weibull model reduces to Schick and Wolverton [28]. In application, one can assume the shape parameter $\beta_1 = \beta_2$.

5.2.3 Change-Point Littlewood Model

Assume F_1 and F_2 are Pareto distribution functions with parameters (λ_1, β_1) and (λ_2, β_2) respectively, given as

$$F_1(t) = \left(1 - (1 + t/\lambda_1)^{\beta_1}\right) \tag{5.2.7}$$

$$F_2(t) = \left(1 - (1 + t/\lambda_2)^{\beta_2}\right) \tag{5.2.8}$$

5.3 Exponential Single Change-Point Model

The NHPP exponential GO model is modified considering single change-point in the testing process. According to the theory of change-point the fault detection rate per remaining fault is varied before and after the change-point, given as

$$b(t) = \frac{\mathrm{d}}{\mathrm{d}t}m(t) \Big/ (a - m(t)) = \begin{cases} b_1 & 0 \le t \le \tau, \\ b_2 & \tau < t \end{cases} \tag{5.3.1}$$

b_1, b_2 are the fault detection rates before and after the change-point. For b_1, $b_2 = b$ the model is equivalent to the GO model. The solution for $m(t)$ under the initial condition at $t = 0$, $m(0) = 0$ and $t = \tau$, $m(t) = m(\tau)$ is

$$m(t) = \begin{cases} a(1 - \mathrm{e}^{-b_1 t}) & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(b_1\tau + b_2(t-\tau))}\right) & t > \tau \end{cases} \tag{5.3.2}$$

and the failure intensity function is defined as

$$\lambda(t) = m'(t) = \begin{cases} ab_1\mathrm{e}^{-b_1 t} & 0 \le t \le \tau, \\ ab_2\mathrm{e}^{-(b_1\tau + b_2(t-\tau))} & t > \tau \end{cases} \tag{5.3.3}$$

The failure intensity function of the change-point models is not continuous, the discontinuity lies at the time point of change. The reliability function of the model for time x given t is defined as

$$\begin{aligned} R(x|t) &= \mathrm{e}^{-(m(t+x)-m(t))} \\ &= \begin{cases} \mathrm{e}^{-a\left(\mathrm{e}^{-b_1 t} - \mathrm{e}^{-b_1(t+x)}\right)} & t \le t + x \le \tau \\ \mathrm{e}^{-a\left(\mathrm{e}^{-b_1 t} - \mathrm{e}^{-\left(b_1\tau + b_2(t+x-\tau)\right)}\right)} & t \le \tau < t + x \\ \mathrm{e}^{-a\left(\mathrm{e}^{-\left(b_1\tau + b_2(t-\tau)\right)} - \mathrm{e}^{-\left(b_1\tau + b_2(t+x-\tau)\right)}\right)} & t > \tau \end{cases} \end{aligned} \tag{5.3.4}$$

This model was proposed by Chang [14]. The authors have suggested using the method of least square to estimate the unknown parameters. The time horizon of change-point is also treated as unknown parameter of the SRGM.

5.4 A Generalized Framework for Single Change-Point SRGM

Recent study on change-point models [23] proposed an approach to develop the change-point models by giving a suitable probability distribution function for the software failure-occurrence times, $F(t)$. Here we describe this generalisation in detail and obtain various exponential and S-shaped change-point models from it. Let $\{N(t),\ t > 0\}$ denote a counting process representing the total number of faults detected up to testing-time t. Then, the probability that m faults are detected up to testing time t is derived as

$$\Pr\{N(t) = m\} = \sum_{n} \binom{n}{m} F(t)^m (1-F(t))^{n-m} \Pr\{N_0 = n\}; \quad m = 0, 1, 2, \ldots \tag{5.4.1}$$

where the random variable N_0 represents the initial number of faults in the system and is finite. If we assume N_0 follows Poisson distribution with mean a, then the counting process $\{N(t),\ t > 0\}$ follows NHPP with mean value function $aF(t)$. That is,

$$\begin{aligned} \Pr\{N(t) = m\} &= \mathrm{e}^{-a}\frac{(aF(t))^m}{m!}\sum_{n}\frac{(a(1-F(t)))^{n-m}}{(n-m)!} \\ &= \frac{(aF(t))^m}{m!}\mathrm{e}^{-aF(t)} \end{aligned} \tag{5.4.2}$$

Equation (5.4.2) implies that an NHPP model can be developed by giving a suitable probability distribution function for the software failure-occurrence times, $F(t)$ with pdf $f(t)$. The single change-point model is described by defining different hazard functions before and after change-point, i.e.

$$b(t) = \begin{cases} b_1(t) = \dfrac{f_1(t)}{1-F_1(t)} & 0 \le t \le \tau, \\ b_2(t) = \dfrac{f_2(t)}{1-F_2(t)} & \tau < t \end{cases} \tag{5.4.3}$$

Using the hazard function defined above the mean value function of the single change-point SRGM is derived from

$$\lambda(t) = m'(t) = af(t) \tag{5.4.4}$$

Equation (5.4.4) can be rewritten as

$$\lambda(t) = (f(t)/1 - F(t))(a - m(t)) \tag{5.4.5}$$

Solving (5.4.5) using (5.4.3) under the initial conditions at $t = 0,\ m(0) = 0$ and at $t = \tau,\ m(t) = m(\tau)$ we get

$$m(t) = \begin{cases} aF_1(t) & 0 \le t \le \tau, \\ a[1 - ((1-F_1(\tau))(1-F_2(\tau))/(1-F_2(\tau)))] & \tau < t \end{cases} \tag{5.4.6}$$

Here $F_i(t)$, $i = 1, 2$ are defined similarly but with different parameters. Kapur et al. [23] used various probability distribution functions such as exponential, Erlang, Logistic, Weibull, Normal, etc. to obtain different single change-point models from the generalization (5.4.6).

5.4.1 Obtaining Exponential SRGM from the Generalized Approach

Assuming the software failure-occurrence time distributions before and after change-points $F_1(t)$, $F_2(t)$ follows exponential distribution i.e.

$$F_1(t) = \left(1 - \mathrm{e}^{-b_1 t}\right), \quad 0 \le t \le \tau \tag{5.4.7}$$

and

$$F_2(t) = \left(1 - \mathrm{e}^{-b_2 t}\right), \quad \tau < t \tag{5.4.8}$$

The mean value function of the change-point SRGM obtained using (5.4.6) is

$$m(t) = \begin{cases} a(1 - \exp(-b_1 t)) & 0 \le t \le \tau, \\ a(1 - \exp(-b_1 \tau - b_2 (t - \tau))) & \tau < t \end{cases} \tag{5.4.9}$$

The software failure-occurrence time distribution for single change-point exponential model can also be derived from [29]

$$F(t) = \begin{cases} F_1(t) = 1 - \mathrm{e}^{-\int_0^t b_1(t)\,\mathrm{d}t} & 0 \le t \le \tau, \\ F_2(t) = 1 - \mathrm{e}^{\left[-\int_0^\tau b_1(t)\,\mathrm{d}t - \int_\tau^t b_2(t)\,\mathrm{d}t\right]} & \tau < t \end{cases} \tag{5.4.10}$$

assuming constant hazard rates, i.e. the mean value function of the SRGM is given as

$$\begin{aligned} m(t) &= aF(t) \\ &= a\{F_1(t)U_1(\tau - t) + F_2(t)U_2(t - \tau)\} \end{aligned} \tag{5.4.11}$$

where $U_1(\cdot)$, $U_2(\cdot)$ are the step functions defined as

$$U_1(x) = \begin{cases} 0 & x < 0 \\ 1 & x \ge 0 \end{cases} \quad \text{and} \quad U_2(x) = \begin{matrix} 0 & x \le 0 \\ 1 & x > 0 \end{matrix} \tag{5.4.12}$$

5.4.2 Obtaining S-Shaped\Flexible SRGM from the Generalized Approach

Change-point models that describe the S-shaped failure curve are also obtainable from the above generalized framework. Using the probability distribution

functions that describe an S-shaped curve in (5.4.6) we can obtain mean value functions of the single change-point SRGM for the various existing S-shaped SRGM and surely many new ones.

If we assume that $F(t)$ is defined by a two stage Erlangian distribution before and after the change point, i.e.

$$F_1(t) = \left(1 - (1 + b_1 t)\mathrm{e}^{-b_1 t}\right), \quad 0 \le t \le \tau \tag{5.4.13}$$

and

$$F_2(t) = \left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right), \quad \tau < t \tag{5.4.14}$$

The mean value function of the change-point SRGM obtained using (5.4.6) is

$$m(t) = \begin{cases} a\left(1 - (1 + b_1 t)\mathrm{e}^{-b_1 t}\right), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1 + b_1 \tau)}{(1 + b_2 \tau)}(1 + b_2 t)\mathrm{e}^{-b_1 \tau - b_2 (t - \tau)}\right) & \tau < t \end{cases} \tag{5.4.15}$$

This model is the S-shaped change-point model for the Yamada et al. [30] delayed S-shaped model. A more flexible S-shaped SRGM is obtained if we assume

$$F_1(t) = \left(1 - \mathrm{e}^{-b_1 t^k}\right), \quad 0 \le t \le \tau \tag{5.4.16}$$

and

$$F_2(t) = \left(1 - \mathrm{e}^{-b_2 t^k}\right), \quad \tau < t \tag{5.4.17}$$

The mean value function of the change-point SRGM obtained using (5.4.6) is

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1 t^k}\right), & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-b_1 \tau^k - b_2 (t^k - \tau^k)}\right) & \tau < t \end{cases} \tag{5.4.18}$$

In this the fault detection rate not only depends on the remaining fault content but depends also on the power function of the testing time. This model has a very special property that for $k = 1$ the model describes an exponential curve, for $k = 2$ it describes a Rayleigh curve while for $k > 2$ the failure distribution is Weibull probability distribution. The shape of the curve changes with the value of k, as such this model can be used for a number of practical applications since the value of k captures the shape of the failure curve.

If we define $F(t)$ by a logistic distribution function i.e.

$$F_1(t) = \frac{1 - \mathrm{e}^{-\mathrm{b_1 t}}}{1 - \beta \mathrm{e}^{-b_1 t}}, \quad 0 \le t \le \tau \tag{5.4.19}$$

and

$$F_2(t) = \frac{1 - \mathrm{e}^{-\mathrm{b}_2 \mathrm{t}}}{1 - \beta \mathrm{e}^{-b_2 t}}, \quad \tau < t \tag{5.4.20}$$

Here in the logistic distribution the shape parameter can be assumed to be equal before and after change-point for the sake of mathematical simplicity. The mean value function of the change-point SRGM is

$$m(t) = \begin{cases} a\left(1 - \dfrac{(1+\beta)\mathrm{e}^{-\mathrm{b}_1 \mathrm{t}}}{1 + \beta \mathrm{e}^{-b_1 t}}\right), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1+\beta)\left(1 + \beta \mathrm{e}^{-\mathrm{b}_2 \tau}\right)}{(1 + \beta \mathrm{e}^{-b_1 \tau})(1 + \beta \mathrm{e}^{-b_2 t})} \mathrm{e}^{-b_1 \tau - b_2 (t - \tau)}\right) & \tau < t \end{cases} \tag{5.4.21}$$

This model is the flexible change-point model for the Kapur and Garg [31] model for error removal phenomenon; it has been proposed by Kapur et al. [20]. The model describes a flexible learning curve and in most cases provides a good estimate of software reliability. It may be noted that the parameter β of the failure time distribution is kept same before and after the change-point. The reason for keeping this parameter constant is only for obtaining a simple mathematical form of the model. For any good reason in any practical situation it can be assumed to vary and the mean value function can be recomputed.

5.4.3 More SRGM Obtained from the Generalized Approach

The distribution the distribution of software failure-occurrence times can be any known probability distribution function, such as Gamma, Normal distributions etc. Each of these distributions has some special characteristics associated with them and in many real life applications can accurately describe the software failure process during the testing process.

Case 1 If the failure occurrence times $T \sim N(\mu, \sigma^2)$ (Normal distribution) i.e.

$$F_i'(t) = g(t; \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\Pi}} e\left(-\frac{(t - \mu_i)^2}{2\sigma_i^2}\right); \quad i = 1, 2 \tag{5.4.22}$$

then the mean value function of the SRGM is

$$m(t) = \begin{cases} a\Phi t; \mu_1, \sigma_1), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1 - \Phi(\tau; \mu_1, \sigma_1))(1 - \Phi(t; \mu_2, \sigma_2))}{(1 - \Phi(\tau; \mu_2, \sigma_2))}\right) & \tau < t \end{cases}$$
$$\Phi(t, \mu, \sigma) = \int_0^t g(x; \mu, \sigma)\, \mathrm{d}x \tag{5.4.23}$$

Case 2 If the failure occurrence times $T \sim \gamma(\alpha_1, \beta_1)$ (Gamma distribution) i.e.

$$F_i'(t) = g(t;\alpha_i,\beta_i) = t^{\alpha_i-1}\frac{\beta_i^{\alpha_i}e^{-\beta_i t}}{\Gamma(\alpha_i)}, \quad t>0;\ i=1,2 \tag{5.4.24}$$

then the mean value function of the SRGM is

$$m(t) = \begin{cases} a\Gamma(t;\alpha_1,\beta_1) & 0\le t\le\tau, \\ a\left(1-\dfrac{(1-\Gamma(\tau;\alpha_1,\beta_1))(1-\Gamma(t;\alpha_2,\beta_2))}{(1-\Gamma(\tau;\alpha_2,\beta_2))}\right) & \tau<t \end{cases} \tag{5.4.25}$$

$$\Gamma(t;\alpha,\beta) = \int_0^t g(x;\alpha,\beta)\,dx$$

5.5 Change-Point SRGM Considering Imperfect Debugging and Fault Complexity

We have studied the importance of considering testing efficiency in measuring the software reliability. As such a change-point SRGM considering the effect of imperfect debugging can provide more accurate measure of reliability. Here first we describe an exponential model [15] which considers only the phenomenon of error generation then an integrated testing efficiency change-point SRGM is developed.

5.5.1 Exponential Imperfect Debugging Model

Along with the assumptions of NHPP and change-point GO model it is assumed that when detected faults are removed at time t, it is possible to introduce new faults with introduction rate $\alpha(t)$

$$\alpha(t) = \begin{cases} \alpha_1 & 0\le t\le\tau, \\ \alpha_2 & \tau<t \end{cases} \tag{5.5.1}$$

Then the set of differential equations that describe the SRGM is given as

$$\frac{dm(t)}{dt} = b(t)(a(t)-m(t)) \text{ where } \frac{da(t)}{dt} = \alpha(t)\frac{dm(t)}{dt} \tag{5.5.2}$$

Solving Eq. (5.5.1) using (5.3.1) under the initial conditions $m(0) = 0$ and $a(0) = a$ we get

$$m(t) = \begin{cases} (a/1-\alpha_1)\left(1-\mathrm{e}^{-b_1(1-\alpha_1)t}\right) & 0 \le t \le \tau, \\ (a/1-\alpha_2)\left(1-\mathrm{e}^{-(b_1(1-\alpha_1)\tau+b_2(1-\alpha_2)(t-\tau)}\right) & \\ \quad + \dfrac{m(\tau)(\alpha_1-\alpha_2)}{1-\alpha_2} & t > \tau \end{cases} \tag{5.5.3}$$

To account the multiple types of the faults in the system (concept of fault complexity) the model can be extended as follows

$$\begin{aligned} m(t) &= \sum_i m_i(t) \\ &= \begin{cases} \left(av_i/1-\alpha_{i,1}\right)\left(1-\mathrm{e}^{-b_{i,1}(1-\alpha_{i,1})t}\right) & 0 \le t \le \tau, \\ \left(\begin{array}{l} \left(av_i/1-\alpha_{i,2}\right) \\ \left(1-\mathrm{e}^{-(b_{i,1}(1-\alpha_{i,1})\tau+b_{i,2}(1-\alpha_{i,2})(t-\tau)}\right) \end{array}\right) & \\ \quad + \dfrac{m(\tau)\left(\alpha_{i,1}-\alpha_{i,2}\right)}{1-\alpha_{i,2}} & t > \tau \end{cases} \end{aligned} \tag{5.5.4}$$

Here the index i denotes the type of fault and v_i, $(b_{i,1}, b_{i,2})$, $(\alpha_{i,1}, \alpha_{i,2})$ are the content proportions, fault detection rate and fault introduction rates of the fault type i in the software, respectively, such that $\sum_i v_i = 1$. The second index represents the parameter value before and after the change-point. This model describes an exponential failure curve and considers only one aspect of testing efficiency i.e. fault generation. We know that the imperfect fault debugging and fault generation together are the measure of testing efficiency and influences the measure of reliability greatly. We now describe here an integrated testing efficiency single change-point SRGM, which also possess the flexible structure that can capture both exponential and S-shaped failure curves.

5.5.2 *Integrated Flexible Imperfect Debugging Model*

Change-point integrated testing efficiency models can be developed for all the integrated testing efficiency models discussed in Sect. 3.5. Here we show how to develop change-point model corresponding to these models for one specific case.

Assuming fault removal rate per additional fault removed is reduced by the probability of perfect debugging and a constant proportion of removed faults are generated during removal, the differential equation describing the removal phenomenon incorporating change-point with imperfect fault debugging and fault generation is given by

$$\frac{dm_r(t)}{dt} = pb(t)(a(t) - m_r(t)) \tag{5.5.5}$$

where

$$b(t) = \begin{cases} b_1/1 + \beta e^{-b_1 t} & 0 \le t \le \tau, \\ b_2/1 + \beta e^{-b_2 t} & t > \tau_J \end{cases} \tag{5.5.6}$$

and

$$a(t) = \begin{cases} a + \alpha_1 m_r(t) & 0 \le t \le \tau \\ a + \alpha_1 m_r(\tau) + \alpha_2 (m_r(t) - m_r(\tau)) & t > \tau \end{cases} \tag{5.5.7}$$

The mean value function of the SRGM obtained from the above equations is

$$m_r(t) = \begin{cases} \dfrac{a}{(1-\alpha_1)}\left[1 - \left(\dfrac{(1+\beta)e^{-b_1 t}}{1+\beta e^{-b_1 t}}\right)^{p(1-\alpha_1)}\right] & 0 \le t \le \tau, \\ \dfrac{a}{(1-\alpha_2)}\left[1 - \left(\left(\dfrac{1+\beta e^{-b_1 \tau}}{1+\beta}\right)^{-p(1-\alpha_1)} \left(\dfrac{1+\beta e^{-b_2 t}}{1+\beta e^{-b_2 \tau}}\right)^{-p(1-\alpha_2)} \right.\right. & \\ \left.\left. e^{-b_1 \tau p(1-\alpha_1) - b_2 (t-\tau) p(1-\alpha_2)}\right)\right] & \\ + \dfrac{a(\alpha_1 - \alpha_2)}{(1-\alpha_1)(1-\alpha_2)}\left[1 - \left(\dfrac{(1+\beta)e^{-b_1 \tau}}{1+\beta e^{-b_1 \tau}}\right)^{p(1-\alpha_1)}\right] & t > \tau \end{cases} \tag{5.5.8}$$

The asymptotic properties of the model are same as SRGM without change point. The difference lies in the mean value function of the SRGM before and after the change-point; however, the intensity function of this model is discontinuous at the change-point. The mean value functions of the failure phenomenon can be obtained from $pm_f(t) = m_r(t)$. This model is due to Sehgal et al. [32]. In this formulation the parameters p and β are taken to be constant and same before and after the change point just for the sake of simplicity and reduce the number of unknown parameters. However they can be taken different, as the parameter p is related to the testing efficiency and changes in this parameter are observed readily with the testing progress due to experience, more removals in the later testing phase, reconstitution of testing and debugging teams and adoption of new testing methods and strategy. Similar changes can be seen in the shape parameter of the logistic function of fault removal rate.

Similarly change-point models can be derived for other testing efficiency models. This is left as an exercise to the readers to derive the change-point model with all parameters different for the above SRGM and change-point models for the other testing efficiency models.

5.6 Change-Point SRGM with Respect to Test Efforts

In the earlier chapters we have discussed various SRGM defined with respect to test efforts. If having a measure of reliability is the major purpose of using an SRGM for practical applications then SRGM developed in respect to time can provide useful information. If however an SRGM is used to measure the effectiveness of resources spent on testing or used in some optimization model for decision-making purpose the models developed accounting the effect of testing resources are more fruitful. Other considerations related to the use of test effort based SRGM have been discussed in the previous chapters.

5.6.1 Exponential Test Effort Models

Exponential test effort single change-point model based on the general assumptions of the NHPP model (GO model) is formulated as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t)\frac{1}{w(t)} = b(t)(a - m(t)) \tag{5.6.1}$$

where

$$b(t) = \begin{cases} b_1 & 0 \le t \le \tau, \\ b_2 & \tau < t \end{cases} \tag{5.6.2}$$

$w(t)$ being the density function of test effort distribution $W(t)$. The mean value function for the failure process is given as

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1(W(t)-W(0))}\right) & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(b_1(W(\tau)-W(0))+b_2(W(t)-W(\tau)))}\right) & \tau < t \end{cases} \tag{5.6.3}$$

or

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1 W^*(t)}\right); \quad W^*(\cdot) = (W(\cdot) - W(0)) & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(b_1 W^*(\tau)+b_2 W(t-\tau))}\right); & \tau < t \end{cases}$$

$$W(t - \tau) = W(t) - W(\tau)) \tag{5.6.4}$$

and the failure intensity function is given as

$$\lambda(t) = \begin{cases} a\, b_1\, w(t)\mathrm{e}^{-b_1 W^*(t)} & 0 \le t \le \tau, \\ a b_2 w(t)\mathrm{e}^{-(b_1 W^*(\tau)+b_2 W(t-\tau))} & \tau < t \end{cases} \tag{5.6.5}$$

Any form of the test effort function discussed in Sect. 2.7 can be used here to describe the distribution of test efforts. Huang [17] has validated this model using logistic and generalized logistic test effort functions.

5.6.2 Flexible/S-Shaped Test Efforts Based SRGM

Development of the flexible test effort models is based on the assumption that the fault detection rate is a function of the time-dependent testing effort consumption function. Mathematically the model is stated as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t)\frac{1}{w(t)} = b(W(t))(a - m(t)) \tag{5.6.6}$$

where the fault detection rate is defined as

$$b(t) = \begin{cases} \dfrac{b_1}{1+\beta \mathrm{e}^{-b_1 W(t)}} & 0 \le t \le \tau, \\ \dfrac{b_2}{1+\beta \mathrm{e}^{-b_2 W(t)}} & \tau < t \end{cases} \tag{5.6.7}$$

The mean value function for the failure process is given as

$$m(t) = \begin{cases} a\left(\dfrac{1-\mathrm{e}^{-\mathrm{b}_1 \mathrm{W}^*(\mathrm{t})}}{1+\beta \mathrm{e}^{-b_1 W(t)}}\right) & 0 \le t \le \tau, \\ a\left[1 - \left(\dfrac{\left(\dfrac{1+\beta \mathrm{e}^{-\mathrm{b}_1 \mathrm{W}(0)}}{1+\beta \mathrm{e}^{-b_1 W(\tau)}}\right)\left(\dfrac{1+\beta \mathrm{e}^{-\mathrm{b}_2 \mathrm{W}(\tau)}}{1+\beta \mathrm{e}^{-b_2 W(t)}}\right)}{\mathrm{e}^{-(b_1 W^*(\tau)+b_2 W(t-\tau))}}\right)\right] & \tau \le t \end{cases} \tag{5.6.8}$$

$W^*(\cdot)$, $W(t-\tau)$ are as defined in (5.6.4). This model was proposed and validated by Kapur et al. [20]. Another form of test effort based model is formulated based on fault detection rate defined as

$$b(t) = \begin{cases} b_1 (W(t))^k & 0 \le t \le \tau, \\ b_2 (W(t))^k & \tau < t \end{cases} \tag{5.6.9}$$

which yields flexible test effort based SRGM. The mean value function for the failure process in this case is given as

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-(1/k+1)b_1 (W(t)^{k+1} - W(0)^{k+1})}\right) & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(1/k+1)\left(b_1\left(W(\tau)^{k+1} - W(0)^{k+1}\right) + b_2\left(W(t)^{k+1} - W(\tau)^{k+1}\right)\right)}\right) & \tau < t \end{cases} \tag{5.6.10}$$

or

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1 (1/k+1) W^*(t)}\right); & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(1/k+1)(b_1 W^*(\tau) + b_2 W(t-\tau))}\right); & \tau < t \end{cases} \tag{5.6.11}$$

$$W^*(\cdot) = \left(W(\cdot)^{k+1} - W(0)^{k+1}\right); \quad W(t-\tau) = \left(W(t)^{k+1} - W(\tau)^{k+1}\right)$$

This model is studied due to Kapur et al. [24]. This model is basically based on the assumption that the failure-occurrence time follows Weibull distribution and

hence the mean value function describes a Weibull probability curve. The models provide a flexible mathematical form of the mean value function. The flexible nature of these models is due to the parameters β, k, respectively. Value of these parameters determines the shape of the curve. For $\beta = 0$ ($k = 1$) the model reduces to GO model type exponential model. Other values of β, $k > 0$ capture the variations of the failure curves.

5.7 SRGM with Multiple Change-Points

First we describe a generalization based on the concept of quasi-arithmetic mean to obtain the mean value function of the NHPP-based SRGM. This generalization is then used to obtain various exponential as well as flexible SRGM with multiple change points [18]. The situation of multiple change points exists in a testing process if changes are observed not only at one point of time rather at various different points of time and the fault detection/removal process between these change points is described by the parameter variation modeling approach, i.e. the process is described by the different set of parameters from a similar distribution.

Quasi-Arithmetic Mean: Let g be a real-valued and strictly monotonic function. Also let x and y be two non-negative real numbers. The quasi-arithmetic mean z of x and y with weights w and $1 - w$ is defined as

$$z = g^{-1}(wg(x) + (1 - w)g(y)); \quad 0 < w < 1 \tag{5.7.1}$$

where g^{-1} is the inverse function of g. We can obtain the weighted arithmetic, weighted geometric and weighted harmonic means from (5.7.1) using $g(x) = x$, $g(x) = 1/x$ and $g(x) = \ln x$, respectively.

Now assume $m(t + \Delta t)$ be equal to the quasi-arithmetic mean of $m(t)$ and a with weights $w(t, \Delta t)$ and $1 - w(t, \Delta t)$, then

$$g(m(t + \Delta t)) = w(t, \Delta t)g(m(t)) + (1 - w(t, \Delta t))g(a); \quad 0 < w(t, \Delta t) < 1 \tag{5.7.2}$$

where g is a real-valued, strictly monotonic and differentiable function. That is,

$$\frac{g(m(t + \Delta t)) - g(m(t))}{\Delta t} = \frac{1 - w(t, \Delta t)}{\Delta t}(g(a) - g(m(t))) \tag{5.7.3}$$

If $\frac{1-w(t,\Delta t)}{\Delta t} \to b(t)$ as $\Delta t \to 0$ then we get the differential equation

$$\frac{\partial}{\partial t} g(m(t)) = b(t)(g(a) - g(m(t))) \tag{5.7.4}$$

Here, $b(t)$ is the fault detection rate per remaining fault. Various NHPP-based SRGM can be obtained from the general equation (5.7.4). The result is summarized in the form of a theorem.

Theorem 1 *Let g be a real-valued, strictly monotonic and differentiable function and*

$$\frac{\partial}{\partial t}g(m(t)) = b(t)(g(a) - g(m(t)))$$

then the mean value function of the NHPP-based SRGM can be obtained from

$$m(t) = g^{-1}(g(a) + g(m(0)) - g(a))\mathrm{e}^{-B(t)} \tag{5.7.5}$$

and $B(t) = \int_0^t b(u)\,\mathrm{d}u$, $g(x) = x$ *and* $k = 1 - \frac{\text{initialcondition}}{a}$, *where initial condition is the value of the mean value function at the boundary point.*

5.7.1 Development of Exponential Multiple Change-Point Model

Based on the weighted arithmetic mean assume $g(x) = x$, $k = 1 - m(0)/a$ and $B(t) = \int_0^t b\,\mathrm{d}u$ then theorem-1 yields the GO model.

$$m(t) = a(1 - \mathrm{e}^{-bt})$$

Following a similar approach the exponential multiple change-point SRGM can be obtained defining

$$b(t) = \begin{cases} b_1 & 0 \le t \le \tau_1, \\ b_2 & \tau_1 < t \le \tau_2, \\ \vdots & \\ \cdot & \\ b_n & \tau_{n-1} < t \end{cases};$$

$$B_i(t) = \int_{\tau_{i-1}}^{t} b_i(u)\,\mathrm{d}u$$

and

$$k_i = 1 - \frac{m_{i-1}(\tau_{i-1})}{a} = \mathrm{e}^{-\sum_{r=1}^{i-1} b_r(\tau_r - \tau_{r-1})}; \quad i = 1, \ldots, n \tag{5.7.6}$$

Using the above definitions the generalized solution of the GO model with multiple change points is

$$m_i(t) = \left\{ a\left(1 - \mathrm{e}^{-\left(b_i(t-\tau_{i-1}) + \sum_{r=1}^{i-1} b_r(\tau_r - \tau_{r-1})\right)}\right); \quad \tau_0 = 0,\ i = 1, \ldots, n \right. \tag{5.7.7}$$

The value of n depends on the number of time points at which changes are observed, which as already discussed can be obtained from the failure data plots and the developers.

5.7.2 Development of Flexible/S-Shaped Multiple Change-Point Model

Following Theorem 1 multiple change-point model can be obtained for the various existing SRGM. Defining $B_i(t)$, k_i as in (5.7.6) and $b_i(t) = \dfrac{b_i}{1 + \beta e^{-b_i t}}$; $i = 1,\ldots, n$, the mean value function of the flexible multiple change-point SRGM is

$$m_i(t) = a\left[1 - \left(\frac{\beta e^{-b_i t} + e^{-b_i(t-\tau_{i-1})}}{1 + \beta e^{-b_i t}} \prod_{r=1}^{i-1} \frac{\beta e^{-b_r t_r} + e^{-b_r(t_r - \tau_{r-1})}}{1 + \beta e^{-b_r t_r}}\right)\right] = 1, \ldots, n. \tag{5.7.8}$$

Here also $\tau_0 = 0$. Following a similar structure various other flexible and S-shaped SRGM can be derived such as multiple change-point models for Yamada delayed S-shaped model [30], Weibull model (Eq. 5.6.9), test effort models, etc. The readers should obtain these models following a similar approach.

So far we have been discussing change-point SRGM. It is said that due to the various reasons that work collectively, changes are observed in the fault detection rate, testing efficiency, etc. These changes bring changes in the failure distribution and parameter variability approach between change-points is used to model the changes. An important fact has been overlooked by these models. Failure occurrence and removal process is described by a numerous factors such as testing environment, testing strategy, complexity and size of the functions under testing, skill, motivation and constitution of the testing and debugging teams, etc. wherein the major role is played by the testing effort expenditure. The reasons that account to bringing variations in the testing process include application of scientific tools and techniques to increase the test coverage, forces from parallel projects, bringing experience and skilled testing professional, distribution of CPU hours, etc. Testing effort distribution during the test phase is affected by most of the factors affecting the testing process and changes in them bring changes in the testing effort consumption rate. Sometimes the testing effort distribution is adjusted to meet the deadline pressures of the project, to discover and remove more faults during the end stage of testing to attain maximum possible reliability. The changes in the testing effort distribution have direct influence on the fault detection and removal and as such cannot be ignored. Now we develop the multiple change-point models to describe the testing effort distribution and using these models we develop the multiple change-point SRGM. Single change-point models can be derived from these models as a special case.

5.8 Multiple Change-Point Test Effort Distribution

5.8.1 *Weibull Type Test Effort Function with Multiple Change Points*

In Chap. 2 we have explained various test effort functions (TEF), these functions are smooth functions and do not consider the varying pattern of testing effort consumption during testing in accordance with the changes brought in the testing strategies, environment, team, etc. to fasten and improve the testing process. Here we develop the modified Weibull type test effort function to describe the varying pattern of test effort consumption by re-parameterizing the differential equation (2.7.9), i.e. assuming the testing effort consumption rate at any time t during the testing process is proportional to the testing resource available at that time and following the procedure of change-point models the test effort function is formulated as

$$\frac{dW(t)}{dt} = v_i(t)[N - W(t)]$$

where

$$v_i(t) = \begin{cases} v_1(t) & 0 \le t \le \tau_1, \\ v_2(t) & \tau_1 < t \le \tau_2, \\ \cdots & \\ v_{n+1}(t) & \tau_n < t \end{cases} \tag{5.8.1}$$

where N is the total testing resource available and $v_i(t)$ is the testing resource consumption rate per remaining effort. Using the above formulation we have the following three non-smooth testing effort functions given n change-points in the testing process.

Case 1: Modified Exponential TEF (METEF) Testing resource consumption rate is defined as

$$v_i(t) = \begin{cases} v_1 & 0 \le t \le \tau_1, \\ v_2 & \tau_1 < t \le \tau_2, \\ \cdots & \\ v_{n+1} & \tau_n < t \end{cases} \tag{5.8.2}$$

The METEF under the initial conditions $W(0) = 0$, $W(\tau = \tau_1) = W(\tau_1)$, …, $W(\tau = \tau_n) = W(\tau_n)$ is

$$W(t) = W_e(t) = \begin{cases} N(1 - e^{-v_1 t}) & 0 \le t \le \tau_1, \\ N\left(1 - e^{-(v_1\tau_1 + v_2(t-\tau_1))}\right) & \tau_1 < t \le \tau_2, \\ \cdots & \\ N\left(1 - e^{-\left(\sum_1^n v_i(\tau_i - \tau_{i-1}) + v_{n+1}(t-\tau_n)\right)}\right) & t > \tau_n \end{cases} \tag{5.8.3}$$

Case 2: Modified Rayleigh TEF (MRTEF) The testing effort consumption rate and MRTEF under the initial conditions $W(0) = 0$, $W(\tau = \tau_1) = W(\tau_1)$, …, $W(\tau = \tau_n) = W(\tau_n)$ are given as

$$v_i(t), W_r(t) = \begin{cases} v_1\, t; & N\left(1 - \mathrm{e}^{-v_1 t^2/2}\right) & 0 \le t \le \tau_1, \\ v_2\, t; & N\left(1 - \mathrm{e}^{-\left(v_1\tau_1^2/2 + v_2/2\left(t^2 - \tau_1^2\right)\right)}\right) & \tau_1 < t \le \tau_2, \\ & \cdots & \\ v_n\, t; & N\left(1 - \mathrm{e}^{-\left(\sum_1^n v_i/2\left(\tau_i^2 - \tau_{i-1}^2\right) + v_{n+1}/2\left(t^2 - \tau_n^2\right)\right)}\right) & t > \tau_n \end{cases} \tag{5.8.4}$$

Case 3: Modified Weibull TEF (MWTEF) The testing effort consumption rate and MRTEF under the initial conditions $W(0) = 0$, $W(\tau = \tau_1) = W(\tau_1)$, …, $W(\tau = \tau_n) = W(\tau_n)$ are given as

$$v_i(t), W_w(t) = \begin{cases} v_1\, c_1\, t^{c_1 - 1}; & N\left(1 - \mathrm{e}^{-v_1 t^{c_1}}\right) & 0 \le t \le \tau_1, \\ v_2\, c_2\, t^{c_2 - 1}; & N\left(1 - \mathrm{e}^{-\left(v_1\tau_1^{c_1} + v_2\left(t^{c_2} - \tau_1^{c_2}\right)\right)}\right) & \tau_1 < t \le \tau_2, \\ & \cdots & \\ v_n\, c_n\, t^{c_n - 1}; & N\left(1 - \mathrm{e}^{-\left(\begin{array}{c}\sum_1^n v_i\left(\tau_i^{c_i} - \tau_{i-1}^{c_i}\right) \\ + v_{n+1}\left(t^{c_{n+1}} - \tau_n^{c_{n+1}}\right)\end{array}\right)}\right) & t > \tau_n \end{cases} \tag{5.8.5}$$

5.8.2 An Integrated Testing Efficiency, Test Effort Multiple Change-Points SRGM

The multiple change-point test effort models are used to develop integrated testing efficiency multiple change-point test effort models. The flexible integrated testing effort model discussed in Sect. 5.5 is extended with respect to the test effort model discussed above assuming n change-points. Let us first recall all the assumptions and considerations that apply to the model.

5.8.2.1 Assumptions

1. Failure observation/fault removal phenomenon is modeled by NHPP.
2. Software failures occur during execution due to faults remaining in the software.
3. As soon as a failure occurs, the fault causing the failure is immediately identified and efforts are made to remove the faults.

4. Weibull type multiple change-point type test effort function describes the consumption of testing resources.
5. The instantaneous rate of fault removal in time $(t,\ t + \Delta t)$ with respect to testing effort is proportional to the mean number of remaining faults in the software.
6. On a removal attempt a fault is removed perfectly with probability p, $0 \le p \le 1$.
7. During the fault removal process, new faults are generated with a constant probability α, $0 \le \alpha \le 1$.

Under the assumptions 1–7 and applying the theory of change-point modeling the differential equation for the SRGM is given by

$$\frac{dm_r(t)}{dt} = \begin{cases} p_1 b_1 (a + \alpha_1 m_r(t) - m_r(t)) w(t) \quad 0 \le t \le \tau_1, \\ p_2 b_2 (a + \alpha_1 m_r(\tau_1) + \alpha_2 (m_r(t) - m_r(\tau_1)) - m_r(t)) w(t) \\ \text{or } p_2 b_2 (a + (\alpha_1 - \alpha_2) m_r(\tau_1) - (1 - \alpha_2) m_r(t)) w(t) \quad \tau_1 < t \le \tau_2, \\ \cdots \\ p_{n+1} b_{n+1} \left(a + \sum_{i=1}^{n} (\alpha_i - \alpha_{i+1}) m_r(\tau_i) - (1 - \alpha_{n+1}) m_r(t) \right) w(t) \\ t > \tau_n \end{cases} \tag{5.8.6}$$

Mean value function (MVF) of the model under the initial conditions $m_r(0) = 0$, $m_r(\tau = \tau_1) = m_r(\tau_1), \ldots, m_r(\tau = \tau_{n-1}) = m_r(\tau_{n-1})$ and $W(0) = 0$, $W(\tau = \tau_1) = W(\tau_1), \ldots, W(\tau = \tau_n) = W(\tau_n)$ is given as

$$m_r(t) = \begin{cases} \dfrac{a}{\bar{\alpha}_1}\left(1 - e^{-q_1 W(t)}\right) & 0 \le t \le \tau_1, \\ \dfrac{1}{\bar{\alpha}_2}\left((a - \bar{\alpha}_1 m(\tau_1))\left(1 - e^{-q_2 (W(t) - W(t_1))}\right) + \bar{\alpha}_2 m(\tau_1)\right) & \tau_1 < t \le \tau_2, \\ \cdots & \\ \dfrac{1}{\bar{\alpha}_{n+1}} \left(\left(\left(a + \sum_{i=1}^{n-1} (\alpha_i - \alpha_{i+1}) m_r(\tau_i) - (1 - \alpha_n) m_r(\tau_n) \right) \left(1 - e^{-q_{n+1}(W(t) - W(t_n))}\right) \right) + (1 - \alpha_{n+1}) m_r(\tau_n) \right) & t > \tau_n \end{cases} \tag{5.8.7}$$

where $\bar{\alpha}_i = 1 - \alpha_i$; $q_i = b_i p_i (1 - \alpha_i)$; $i = 1, 2, \ldots, n$.

The study is due to Gupta et al. [4]. Such a model is very useful for the reliability analysis as the measure of reliability is computed considering the distribution of testing efforts, influence of the testing efficiency and the changes of the testing process.

5.9 A Change-Point SRGM with Environmental Factor

The study of reliability models with change points reveals that a great improvement in the accuracy of evaluation of software reliability is achieved with the use of change-point models as it considers the more realistic situations of the testing process. The models describe the difference of testing environments before and after the change point using different fault detection rates, while traditional models have ignored such differences completely. In fact, there are both differences and links between the fault detection rates before and after the change point. Software testing is an integrated and continuous process. The software testing process consists of several testing stages, including unit testing, integration testing and system testing. At the stages of testing, the test teams and the operating systems are similar. So, the fault detection rates before and after the change point should have some links with each other because of the similarity of the environments and these links can be described using the environmental factors. Environmental factors that profile the software development process have much impact on software reliability, which is studied by some researchers [33, 34], who identify six factors that have the most significant impact on software reliability including software complexity, programmer skill, testing effort, testing coverage, testing environment and frequency of program specification change. Environmental factors include many other important factors that affect software reliability, which need to be considered and incorporated into the software reliability assessment. These environmental factors can be used to associate the fault detection rate before the change point with fault detection rates after the change point. In fact, the environments that respective phases experience during the software testing process are also different. In order to quantify the environment mismatch due to the change-point problems of testing, in a study an environmental factor was proposed [35] which is used to describe the differences between the system test environment and the field environment. This factor is defined as

$$k_i = \frac{\bar{b}_{\text{test}}}{\bar{b}_{\text{field}}} \tag{5.9.1}$$

This factor is used to link the fault detection rates of the testing and the operational phases. $\bar{b}_{\text{test}}, \bar{b}_{\text{field}}$ respectively represent the long-term average per fault failure rate during the system test and the field. This factor is assumed to remain constant. From the aspect of the software testing process, the testing phase is based on a testing profile, developed test cases and uses various test strategies. Different test cases have different failure detection capability. At any of the testing phases, firstly the testers are observed to run the test cases with strong testing capability and high percentage of coverage to improve the testing speed and efficiency, which will lead to reduction of the FDR. If the testing transfers to a new phase, the FDR still decreases similarly. It is very difficult to ensure that the two FDRs decrease in a same proportion during the testing phases. Therefore, for better description of the impact of environment on the FDR, a function varying with time

should be used to describe environmental factors. More precisely, the FDR is used to measure the effectiveness of fault detection of test techniques and test cases. Four kinds of FDR functions during software testing are defined in the literature.

1. Constant [36].
2. An increasing function with respect to the testing time [37].
3. A decreasing function with respect to the testing time [30].
4. First increasing and then decreasing function with respect to the testing-time [38].

If same SRGM is used before and after the change-points, the relationship between time and FDRs are as shown in Figs. 5.1, 5.2, 5.3 and 5.4. The figures clearly illustrates that the environmental factor is a constant in the first case and may be a variable in the other three cases. Thus, more generally, the environmental factor should be defined as a function of time. Here $b_{bf}(t)$ denotes the FDR before the change point and $b_{af}(t)$ is the FDR after the change point.

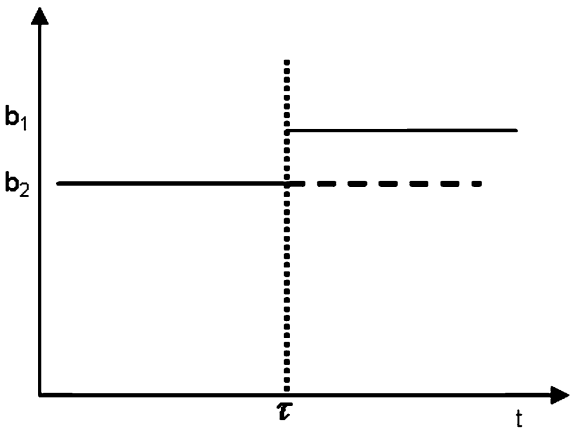


Fig. 5.1 FDR constant with respect to time

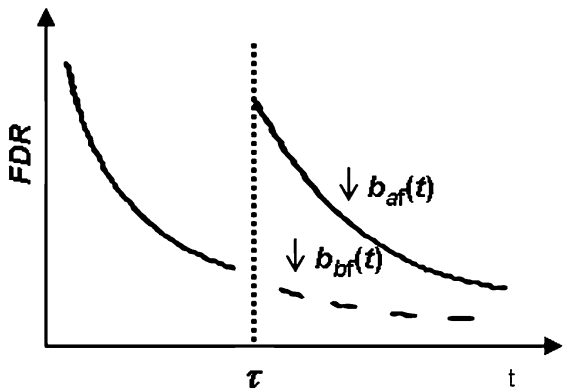


Fig. 5.2 FDR increasing function with respect to the testing time

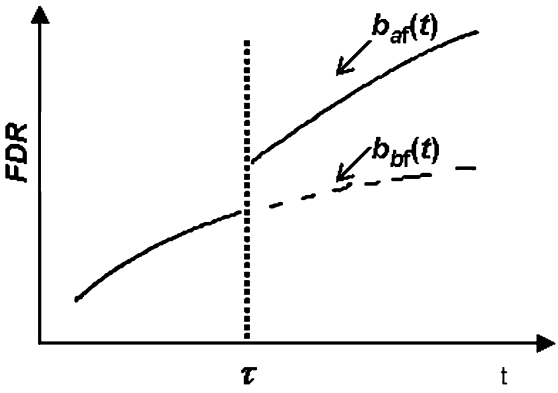


Fig. 5.3 FDR decreasing function with respect to the testing time

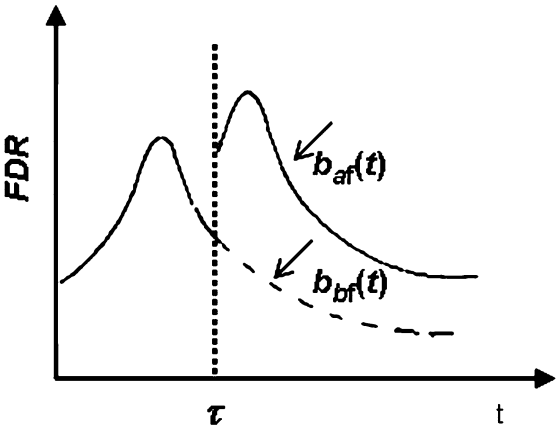


Fig. 5.4 First increasing and then decreasing FDR

The time-dependent environmental factor is defined as

$$k(t) = \frac{b_{\mathrm{bf}}(t)}{b_{\mathrm{af}}(t)}; \quad t \in (\tau, +\infty] \tag{5.9.2}$$

and the average time-varying environmental factor is defined as

$$\bar{k}(t) = \frac{\bar{b}_{\mathrm{bf}}(t)}{\bar{b}_{\mathrm{af}}(t)}; \quad t \in (\tau, +\infty] \tag{5.9.3}$$

where $b_{\mathrm{bf}}(t)(\bar{b}_{\mathrm{bf}}(t)), b_{\mathrm{af}}(t)(\bar{b}_{\mathrm{af}}(t))$ denote the FDRs (average) before and after the change-point τ. Assume that the testing ends at t_{end}. The expected number of faults detected and removed by time τ is $m(\tau)$ and the FDR before the change-point is $b_{\mathrm{bf}}(t)$. After change point, the expected and actual number of faults detected and removed by time t is $m(t)$ and $N(t)$, respectively.

The residual number of faults after the change-point is $\bar{N}(t) = a - N(t)$. $\bar{N}(t)$ can be obtained by replacing a with its estimate by applying all the failure data to a similar type of SRGM without change-point, such as GO model for an exponential failure curve. This also gives a measure of $b_{\mathrm{bf}}(t)$. The failure intensity after the change-point (case of GO model) is given as

$$\lambda(t) = b_{\mathrm{af}}(t)(a - m(t)) \tag{5.9.4}$$

The following equation can be used to calculate average failure intensity

$$\lambda(t) = \frac{N(t_i) - N(t_{i-1})}{t_i - t_{i-1}} \tag{5.9.5}$$

Now replacing $m(t)$ with $N(t)$, $\lambda(t)$ with $\bar{\lambda}(t)$, the average FDR is calculated as

$$\bar{b}_{\mathrm{af}}(t_i) = \frac{\lambda(t_i)}{a - N(t_i)} \tag{5.9.6}$$

Discrete and average time-varying environmental factors of $\bar{k}(t)$ can thus be calculated as

$$\bar{k}(t_i) = \frac{\bar{b}_{\mathrm{bf}}(t_i)}{\bar{b}_{\mathrm{af}}(t_i)}; \quad t_i \in (\tau, t_{\mathrm{end}}] \tag{5.9.7}$$

Zhao et al. [39] carried out a study on two data sets reported by Ohba [40] and Musa et al. [41]. Firstly plotting the data failure trends was determined. For Ohba data set GO model, Yamada delayed S-shaped model [30] and logistic growth curve $\left(m(t) = a/\left(1 + \beta \mathrm{e}^{-bt}\right)\right)$ are fitted, while on the Musa's data observing the S-shaped trend only the S-shaped models (delayed S-shaped and logistic growth curves) were fitted. Comparison result of parameter estimation showed that the logistic curve fitted best to both of the data. Using (5.9.4) $b_{\mathrm{bf}}(t)$ is obtained as

$$b_{\mathrm{bf}}(t) = \frac{a}{1 + \beta \mathrm{e}^{-bt}} \tag{5.9.8}$$

$b_{\mathrm{bf}}(t)$ of Eq. (5.9.8) is the non-decreasing S-shape, which denotes the testers' learning process. The learning is closely related to the changes in the efficiency of testing during a testing phase. The idea is that in organizations that have advanced software processes, testers might be allowed to improve their testing process as they learn more about the product. This could result in a fault detection rate increase monotonically over the testing period. As the testing continues, the increase of FDR becomes slow gradually, the failure intensity of software will decrease significantly, the effectiveness of the testing will be lowered, and thus the tester will adopt new testing technologies and measures to improve the number of failures detected within a unit time, therefore the change point is generated. Thus $\bar{b}_{\mathrm{bf}}(t)$ can be approximately replaced by the FDR at the maximum level $\bar{b}_{\mathrm{bf}}(t)$ before the change-point of testing. While $\bar{b}_{\mathrm{af}}(t)$, $\bar{k}(t)$ are derived using (5.9.6) and

(5.9.7). From the above experiments of two data sets, the approximately decreasing trends of $\bar{k}(t)$ are derived. This is due to the fact that as the testing proceeds, the effective use of testing strategies and tools of non-random testing makes the average FDR after the change point of testing approximately non-decreasing, thus the average environmental factor is decreasing with time. The approximately decreasing trend of $\bar{k}(t)$ can be described as

$$\bar{k}(t) = B\mathrm{e}^{-\delta t} \tag{5.9.9}$$

Now an NHPP-based change-point model assuming perfect debugging environment is derived on the basic assumptions of the change-point models and assuming that before the change point of testing, the fault detection rate captures the learning process of software testers; and after the change point of testing, the fault detection rate is the integrated result of environmental effects and the FDR before the change-point. The mean value function before and after the change point are derived as

$$\frac{\mathrm{d}}{\mathrm{d}t}m(t) = \begin{cases} b_{bf}(t)(a - m(t)) & 0 \le t \le \tau \\ b_{af}(t)(a - m(t)) & t > \tau \end{cases} \tag{5.9.10}$$

Using the initial conditions at $t = 0$, $m(t) = 0$ and $t = \tau$, $m(t) = m(\tau)$ and approximating $b_{af}(t)$ using $b_{af}(t) = \dfrac{\bar{b}_{bf}}{\bar{k}(t)} = \dfrac{\bar{b}_{bf}}{B\mathrm{e}^{-\delta t}}$ the mean value function of the SRGM before and after the change-point is given as

$$m(t) = \begin{cases} \dfrac{a}{1 + \beta \mathrm{e}^{-bt}} & 0 \le t \le \tau \\ (a - m(\tau))\left(1 - \mathrm{e}^{-B^* t}\right) & t > \tau \end{cases} \tag{5.9.11}$$

where

$$B^*(t) = \int_{\tau}^{t_{end}} b_{af}(t)\,\mathrm{d}t = \int_{\tau}^{t_{end}} \frac{\bar{b}_{bf}}{B\mathrm{e}^{-\delta t}}\,\mathrm{d}t = \frac{\bar{b}_{bf}\left(\mathrm{e}^{-\delta t_{\mathrm{end}}} - \mathrm{e}^{-\delta \tau}\right)}{B\delta} \quad t > \tau \tag{5.9.12}$$

Using (5.9.12) the mean value function of the SRGM is given as

$$m(t) = \begin{cases} \dfrac{a}{1 + \beta \mathrm{e}^{-bt}} & 0 \le t \le \tau \\ (a - m(\tau))\left(1 - e\left(-\bar{b}_{bf}\left(\mathrm{e}^{-\delta t_{end}} - \mathrm{e}^{-\delta \tau}\right)\right)/B\delta\right) + m(\tau) & t > \tau \end{cases} \tag{5.9.13}$$

The above change-point SRGM describes an S-shaped failure curve considering the environmental factors for determining the FDR before and after the change-points. Change-point models find very interesting application which is called testing effort control problem during software testing. In the next section we describe how to carry out a testing effort control problem [20] on the testing effort based change-point flexible SRGM (Eq. 5.6.8).

5.10 Testing Effort Control Problem

During testing, often the developer management is not satisfied with the progress of the testing and the growth of the failure growth curve or it may happen that the reliability level achievable with the current testing level does not match with the desired level upto the scheduled delivery time. Then there arises need for employing additional testing efforts in terms of new techniques, testing tools, more manpower so as to remove more faults than what could be possibly achieved with the current level of testing efforts in a prespecified time interval. This is a trade of problem between the aspiration level of reliability and the testing resources. This analysis gives an insight into the current level of progress in testing and later on helps in the estimation of extra efforts/cost required to achieve the aspiration level.

Let us consider the case when testing has been in progress for time T_1 and the number of faults removed by time T_1 is $m(T_1)$. Let T_2 be the release time of the product in the market. Then by time T_2, the number of faults removed will rise to $m(T_2)$ if testing is continued and similar efforts are put. The level $m(T_2)$ may or may not coincide with the level to be achieved by the release time (say m^*). To accelerate testing and increase the efficiency of the testing and debugging teams it is required to put extra resources in terms of additional man-hours, new testing techniques, tools and more skilled testing personnel. Now the question arises how much additional efforts above the current level need to be employed to achieve the level m^*. In the testing effort control problem we estimate the requirement for additional efforts for the aspiration level to achieve.

First using the actual failure data for time $(0, T_1)$, we estimate the parameters of the SRGM (5.6.8). Using the estimated values of the parameters the number of faults, which can be removed, by time T_2 is $m(T_2)$. If $m^* < m(T_2)$, then the current level of testing is sufficient to reach the target reliability level. The team has just to sustain the current level following the similar testing environment as earlier and the product is expected to be ready for the delivery without any urgency. But if $m^* > m(T_2)$, then there is the urgency of accelerating the fault removal rate by increasing efficiency of the testing. The aim is to estimate the requirement for additional testing efforts for the time interval $[T_1, T_2)$ so as to remove $(m^* - m(T_1))$ faults by time T_2. In this case time T_1 is a change point as from this point onwards the testing team has to follow a different, advanced set of test efforts, tools and strategy to achieve removal of $(m^* - m(T_1))$ $(>(m(T_2) - m(T_1)))$ number of faults in the time period $[T_1, T_2)$. The change point will deviate the growth curve at an accelerating pace. It may be noted that the time point T_1 may not be that first change point one or more changes point can occur before time T_1. We assume that a change-point have occurred before time T_1 $(T_1 > \tau)$. Then in this case if the current level of testing efficiency is maintained, then the number of faults removed by time T_2 i.e. $m(T_2)$ is given as

$$m(T_2) = a\left[1 - \left(\frac{1+\beta e^{-b_1 W(0)}}{1+\beta e^{-b_1 W(\tau)}}\right)\left(\frac{1+\beta e^{-b_2 W(\tau)}}{1+\beta e^{-b_2 W(T_2)}}\right) e^{-(b_1 W^*(\tau)+b_2 W(T_2-\tau))}\right] \tag{5.10.1}$$

Let $m^* = m(T_2) + m(T_2 - T_1)$. Here $m(T_2 - T_1)$ is the additional number of faults that need to be removed to reach m^* by time T_2. Let $W(T_1)$ be the cumulative level of the testing effort used for time $(0, T_1)$ and $W(T_2)$ be the cumulative level of the testing effort used in time $[T_1T_2)$ if testing is continued with the same pace as up to the time $(0, T_1)$. Let $W(T_2 - T_1)$ be the additional amount of efforts required to remove $m(T_2 - T_1)$ faults during interval $[T_1, T_2)$. This control problem can be presented graphically as in Fig. 5.5.

For $t > T_1$ the removal process can be represented by the following differential equation

$$\frac{dm(t)}{dt} \Big/ w(t) = \frac{b_2}{1 + \beta e^{-b_2 W(t)}}(a - m(T_1) - m(t)) \tag{5.10.2}$$

Let us define $a_1 = a - m(T_1)$, then the above differential equation can be written as

$$\frac{dm(t)}{dt} \Big/ w(t) = \frac{b_2}{1 + \beta e^{-b_2 W(t)}}(a_1 - m(T_1) - m(t)) \tag{5.10.3}$$

$$m(t) = m(T_1) + a_1 \left(\frac{1 - e^{-b_2 W(t)}}{1 + \beta e^{-b_2 W(t)}} \right) \tag{5.10.4}$$

If the desirable level for the fault removal is m^*, then the requirement for the additional efforts can be generated by the following expression

$$m^* = m(T_1) + a_1 \left(\frac{1 - e^{-b_2 W^{\#}}}{1 + \beta e^{-b_2 W^{\#}}} \right) \tag{5.10.5}$$

With the estimated values of parameters a_1, b_2, β and $m(T_1)$, the above expression can be solved to find the value of $W^{\#}$ corresponding to different values of m^*.

Here

$$W^{\#} = W(T_2) - W(T_1) \tag{5.10.6}$$

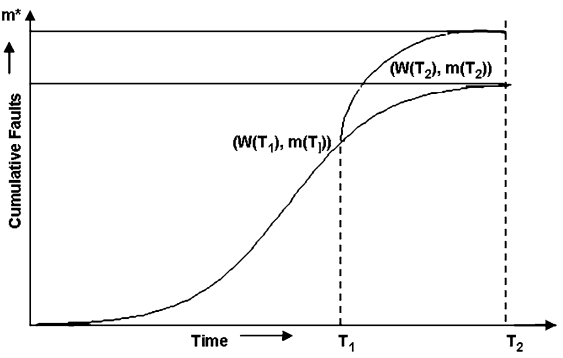


Fig. 5.5 Testing effort control problem

where $W^{\#}$ represents the amount of additional efforts required for the time interval (T_1, T_2) to remove m^* faults from the software.

5.11 Data Analysis and Parameter Estimation

Before discussing the change-point models we stated the fact that these models describe the changes observed in the testing process during testing, as a result these models have better estimating and predictive power than those without their change-point counterparts. This section of the chapter proves this fact. Here we have established the validity of models from both categories and estimated the parameters of these models. A comparison is then drawn on their estimating and predictive capabilities. Several models have been selected describing the various aspects of the testing process, such as uniform and non-uniform operational profiles, testing efficiency model, test effort based model and models based on statistical Weibull, Normal and Gamma distributions.

5.11.1 Models with Single Change-Point

Failure Data Set

The data set has been cllected during 19 weeks of testing of a real time command and control system and 328 faults were detected during the testing [40]. Analysis of the graphical plot of this data set depicts a change-point at 6th week of testing.

Following models have been chosen for data analysis and parameter estimation.

Model 1 (M1) Exponential GO model [36] (refer Sect. 2.3.1)

$$m(t) = a(1 - \mathrm{e}^{-bt})$$

Model 2 (M2) Exponential change-point model [14]

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1 t}\right) & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-(b_1\tau + b_2(t-\tau))}\right) & t > \tau \end{cases}$$

Model 3 (M3) Yamada S-shaped model [30] (refer Sect. 2.3.4)

$$m_f(t) = a(1 - \mathrm{e}^{-bt})$$

Model 4 (M4) S-shaped change-point model [23]

$$m(t) = \begin{cases} a\left(1 - (1 + b_1 t)\mathrm{e}^{-b_1 t}\right), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1 + b_1\tau)}{(1 + b_2\tau)}(1 + b_2 t)\mathrm{e}^{-b_1\tau - b_2(t-\tau)}\right) & \tau < t \end{cases}$$

Model 5 (M5) Flexible SRGM [31] (refer Sect. 2.3.4)

$$m(t) = a\left[\frac{1 - \mathrm{e}^{-\mathrm{bt}}}{1 + \beta \mathrm{e}^{-bt}}\right]$$

Model 6 (M6) Flexible change-point model [20]

$$m(t) = \begin{cases} a\left(1 - \dfrac{(1+\beta)\mathrm{e}^{-\mathrm{b_1 t}}}{1 + \beta \mathrm{e}^{-b_1 t}}\right), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1+\beta)\left(1 + \beta \mathrm{e}^{-\mathrm{b_2}\tau}\right)}{(1 + \beta \mathrm{e}^{-b_1 \tau})(1 + \beta \mathrm{e}^{-b_2 t})} \mathrm{e}^{-b_1 \tau - b_2 (t-\tau)}\right) & \tau < t \end{cases}$$

Model 7 (M7) Weibull model [23]

$$a\left(1 - \mathrm{e}^{-bt^k}\right)$$

Model 8 (M8) Weibull change-point model [23]

$$m(t) = \begin{cases} a\left(1 - \mathrm{e}^{-b_1 t^k}\right), & 0 \le t \le \tau, \\ a\left(1 - \mathrm{e}^{-b_1 \tau^k - b_2 (t^k - \tau^k)}\right) & \tau < t \end{cases}$$

Model 9 (M9) Normal distribution change-point model [23]

$$m(t) = \begin{cases} a\Phi t; \mu_1, \sigma_1), & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1 - \Phi(\tau; \mu_1, \sigma_1))(1 - \Phi(t; \mu_2, \sigma_2))}{(1 - \Phi(\tau; \mu_2, \sigma_2))}\right) & \tau < t \end{cases}$$

$$\Phi(t, \mu, \sigma) = \int_0^t g(x; \mu, \sigma)\, \mathrm{d}x$$

Model 10 (M10) Gamma distribution change-point model [23]

$$m(t) = \begin{cases} a\Gamma(t; \alpha_1, \beta_1) & 0 \le t \le \tau, \\ a\left(1 - \dfrac{(1 - \Gamma(\tau; \alpha_1, \beta_1))(1 - \Gamma(t; \alpha_2, \beta_2))}{(1 - \Gamma(\tau; \alpha_2, \beta_2))}\right) & \tau < t \end{cases}$$

$$\Gamma(t; \alpha, \beta) = \int_0^t g(x; \alpha, \beta)\, \mathrm{d}x$$

Model 11 (M11) Exponential change-point imperfect debugging model [15]

$$m(t) = \begin{cases} (a/1 - \alpha_1)(1 - \mathrm{e}^{-b_1 (1-\alpha_1) t}) & 0 \le t \le \tau, \\ (a/1 - \alpha_2)\left(1 - \mathrm{e}^{-(b_1 (1-\alpha_1)\tau + b_2 (1-\alpha_2)(t-\tau)}\right) & t > \tau \\ + \dfrac{m(\tau)(\alpha_1 - \alpha_2)}{1 - \alpha_2} \end{cases}$$

Table 5.1 Estimation results for model 1 to model 12

Model	Estimated parameters						Comparison criteria	
	a	b, b_1, μ_1, β_1	b_2, μ_2, β_2	k, β	σ_1, α_1	σ_2, α_2	MSE	R^2
M1	761	0.032	–	–	–	–	158.71	0.986
M2	423	0.055	0.098	–	–	–	86.62	0.993
M3	374	0.198	–	–	–	–	188.60	0.984
M4	393	0.196	0.175	–	–	–	192.03	0.984
M5	382	0.179	–	2.886	–	–	98.26	0.992
M6	405	0.079	0.122	0.461	–	–	91.26	0.993
M7	428	0.036	–	1.284	–	–	121.79	0.990
M8	429	0.058	0.112	0.951	–	–	91.16	0.993
M9	339	–0.097	9.355	–	5.550	4.536	52.79	0.996
M10	411	5.240	6.012	–	1.920	2.321	266.85	0.965
M11	451	0.051	0.085	–	0.007	0.001	106.33	0.994
M12	438	0.018	0.033	–	–	–	252.43	0.993

Table 5.2 Estimation results for test effort functions

Test effort function	Estimated parameters			Comparison criteria	
	α	v	c, β	MSE	R^2
Exponential	590	0.004	–	25.71	0.99
Rayleigh	49	0.014	–	26.49	0.974
Weibull	180	0.009	1.192	28.22	0.996
Logitic	55	0.226	13.033	1.93	0.992

Model 12 (M12) Exponential test effort model [17]

$$\lambda(t) = \begin{cases} a\, b_1\, w(t) \mathrm{e}^{-b_1 W^*(t)} & 0 \le t \le \tau, \\ a\, b_2\, w(t) \mathrm{e}^{-(b_1 W^*(\tau) + b_2 W(t-\tau))} & \tau < t \end{cases}$$

$$W^*(\cdot) = (W(\cdot) - W(0)),\ W(t-\tau) = (W(t) - W(\tau))$$

The parameters of models M1–M13 are estimated using the regression module of SPSS. Results of estimation for SRGM have been tabulated in Table 5.1 and those for the test effort functions are tabulated in Table 5.2. Figures 5.6, 5.7, 5.8 and 5.9 show the comparison of goodness of fit and future predictions for the change-point models and their non-change-point counterparts for the exponential, S-shaped, flexible and Weibull SRGM, respectively. Goodness of fit curve and future predictions for all change-point models are shown in Fig. 5.10. Fitting of test effort function and the test effort based SRGM is shown in Figs. 5.11 and 5.12, respectively,

From the estimation results we can see that the change-point models provide better fit as compared to their without change-point counterparts. The mean square error of the GO model with change-point is 86.62 while that for the GO model it is 158.71, the difference between the two MSE is 72.09, which is quite big. Similarly we see the same results for other models also, except for models M3 and M4.

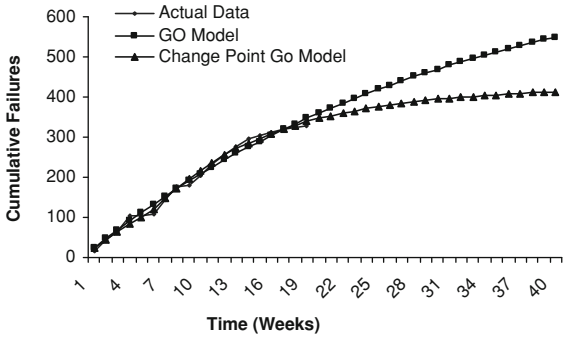


Fig. 5.6 Goodness of fit curve for models 1 and 2

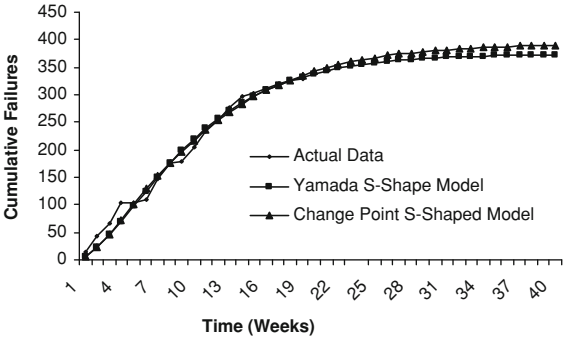


Fig. 5.7 Goodness of fit curve for models 3 and 4

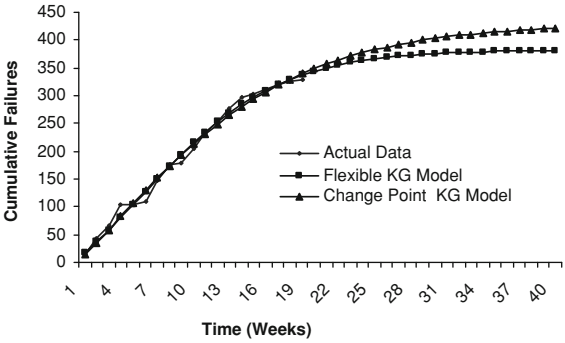


Fig. 5.8 Goodness of fit curve for models 5 and 6

The normal distribution based SRGM provided the best fit for this data set with MSE value 52.79 and R^2 value 0.996.

5.11.2 Models with Multiple Change Points

In this section we show an application of multiple change-point SRGM. Such a case is observed when the testing process is reviewed frequently and the reliability

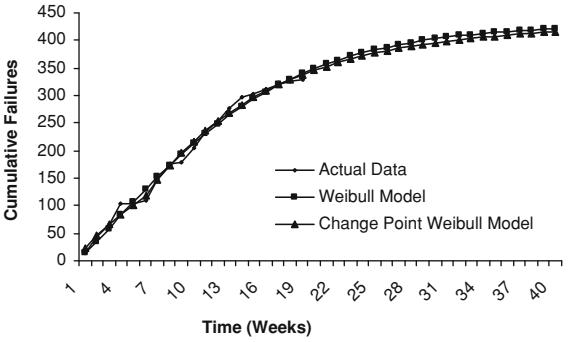


Fig. 5.9 Goodness of fit curve for models 7 and 8

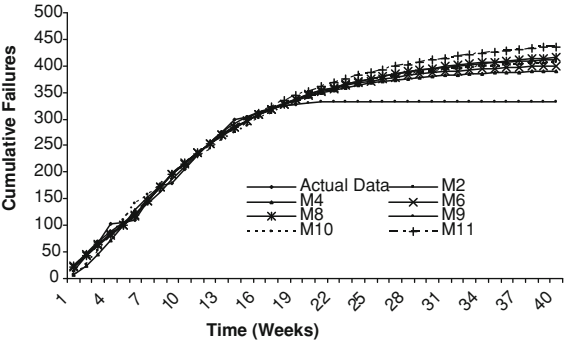


Fig. 5.10 Goodness of fit curve for all change-point models

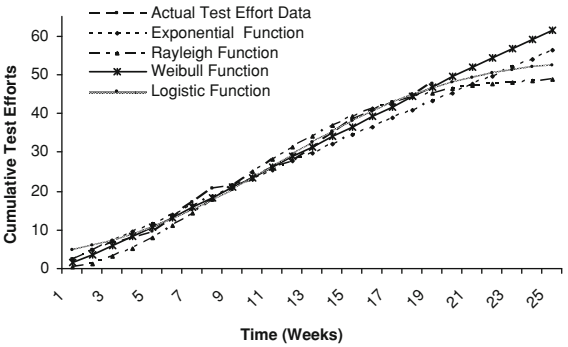


Fig. 5.11 Goodness of fit curve for test effort functions

growth curve changes shape due to the changes made in the testing process at various review points.

Failure Data Set

We continue with the data chosen in the previous section as they can facilitate the comparison of the without change-point, single change-point and multiple

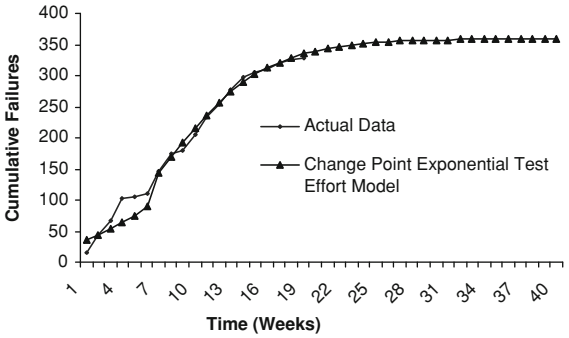


Fig. 5.12 Goodness of fit curve for test effort based model (M12)

Table 5.3 Estimation results for test effort functions

Model	Estimated parameters				Comparison criteria	
	a	b_1	b_2	b_3	MSE	R^2
M13	428	0.057	0.0933	0.0893	112.12	0.991

change-point models. Analysis of the graphical plot of the data set depicts that after the change-point at 6th week of testing another changing pattern is observed in the data at the 9th week of testing. We assume the observed failure data have two change-points at the 6th and 9th weeks respectively.

Following model has been chosen for data analysis and parameter estimation.

Model 13 (M13) Multiple change-point exponential GO model [18]

$$m_i(t) = \left\{ a\left(1 - \mathrm{e}^{-b_i(t-\tau_{i-1})+\sum_{r=1}^{i-1} b_r(\tau_r-\tau_{r-1})}\right); \quad \tau_0 = 0,\ i = 1,\ldots,n \right.$$

Comparing the estimation results of GO model without change-point, with single change-point and two change-points (Table 5.3 and Fig. 5.13) we can see that the GO exponential model with single change-point provides a better fit with MSE 86.62 and R^2 value 0.993. The poor fitting of model with two change-points suggests that data have only one point at which changes are observed in the testing process. The occurrence of another shift in the data may not be significant. Although if the multiple change-points exist in the data, the multiple change-point models provide better fit than single change-point and without change-point models.

5.11.3 Change-Point SRGM Based on Multiple Change-Point Weibull Type TEF

Section 5.8 describes the development of the multiple change-point models based on modified Weibull type testing effort functions. These testing effort functions

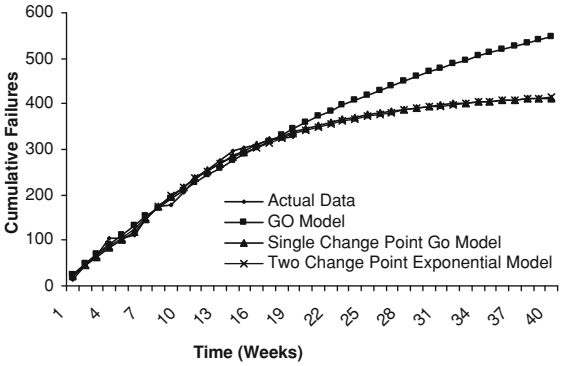


Fig. 5.13 Goodness of fit curve of exponential model with one, two and no change points

have been modeled accounting the changes observed at the one or more change-points observed in the testing process. As already discussed the progress of the testing process largely depends on testing efforts, which are reviewed and adjusted during testing to control the progress. In order to incorporate these changes the traditional test effort functions defined in the literature are modified. In this section we demonstrate an application of modified test effort functions and SRGM developed based on them.

Failure Data Set

This data set is from Brooks and Motley [42]. The failure data set is for a radar system of size 124 KLOC (Kilo Lines of Code) tested for 35 weeks (1846.92 CPU hours) in which 1,301 faults were detected. This application is based on single change point both in the test effort and failure data. The single change point is observed at the time point 7 weeks, i.e. $\tau_1 = 7$. In this application we have also shown the comparison of the estimating and predicting powers of with and without change-point test effort functions as well as the SRGM.

Traditional and Modified Weibull type test effort functions selected for application are

Model 14 (M14) Exponential TEF [43]

$$W(t) = W_e(t) = N(1 - e^{-vt})$$

Model 15 (M15) Modified exponential TEF [4]

$$W(t) = W_e(t) = \begin{cases} N(1 - e^{-v_1 t}) & 0 \le t \le \tau_1, \\ N\left(1 - e^{-(v_1\tau_1 + v_2(t-\tau_1))}\right) & t > \tau_1 \end{cases}$$

Model 16 (M16) Rayleigh TEF [43]

$$W_r(t) = N\left(1 - e^{-vt^2/2}\right)$$

Model 17 (M17) Modified Rayleigh TEF [4]

$$W_r(t) = \begin{cases} N\left(1 - \mathrm{e}^{-v_1 t^2/2}\right) & 0 \le t \le \tau_1, \\ N\left(1 - \mathrm{e}^{-\left(v_1 \tau_1^2/2 + v_2/2\left(t^2 - \tau_1^2\right)\right)}\right) & t < \tau_1, \end{cases}$$

Model 18 (M18) Weibull TEF [43]

$$W_w(t) = N\left(1 - \mathrm{e}^{-vt^c}\right)$$

Model 19 (M19) Modified Weibull TEF [4]

$$W_w(t) = \begin{cases} N\left(1 - \mathrm{e}^{-v_1 t^{c_1}}\right) & 0 \le t \le \tau_1, \\ N\left(1 - \mathrm{e}^{-\left(v_1 \tau_1^{c_1} + v_2\left(t^{c_2} - \tau_1^{c_2}\right)\right)}\right) & t > \tau_1, \end{cases}$$

and the SRGM selected for application are

Model 20 (M20) Exponential test effort based SRGM incorporating testing efficiency [4]

$$m_r(t) = (a/(1-\alpha))\left(1 - \mathrm{e}^{-bp(1-\alpha)W(t)}\right)$$

Model 21 (M21) Exponential change-point test effort based SRGM with multiple change-points incorporating testing efficiency [4]

$$m_r(t) = \begin{cases} (a/(1-\alpha_1))\left(1 - \mathrm{e}^{-b_1 p_1 (1-\alpha_1) W(t)}\right) & 0 \le t \le \tau_1, \\ \left(a/\begin{pmatrix}(1-\alpha_1)\\(1-\alpha_2)\end{pmatrix}\right)\begin{pmatrix}(1-\alpha_2) - (\alpha_1 - \alpha_2)\mathrm{e}^{-b_1 p_1 (1-\alpha_1) W(t_1)} \\ - (1-\alpha_1)\mathrm{e}^{-\begin{pmatrix} b_1 p_1 (1-\alpha_1) W(t_1) \\ + b_2 p_2 (1-\alpha_2)(W(t) - W(t_1)) \end{pmatrix}}\end{pmatrix} & t > \tau_1 \end{cases}$$

Estimated values of parameters of the test effort functions are tabulated in Table 5.4. Goodness of fit curves for the test effort exponential, Rayleigh and Weibull functions and their respective change-point forms are shown in Figs. 5.14, 5.15, and 5.16.

Table 5.4 Estimation results for with- and without-change-point test effort functions

Model	Estimated parameters					Comparison criteria	
	N	v, v_1	v_2	c_1	c_2	MSE	R^2
M14	2,679	0.0226	–	–	–	56,629.76	0.8492
M15	2,956	0.0073	0.0264	–	–	29,793.66	0.9384
M16	2,873	0.0018	–	–	–	1,200.86	0.9983
M17	2,791	0.0016	0.0018	–	–	1,069.02	0.9983
M18	2,692	0.0008	2.0650	–	–	999.08	0.9983
M19	2,809	0.0002	0.0009	2.695	1.999	586.94	0.9984

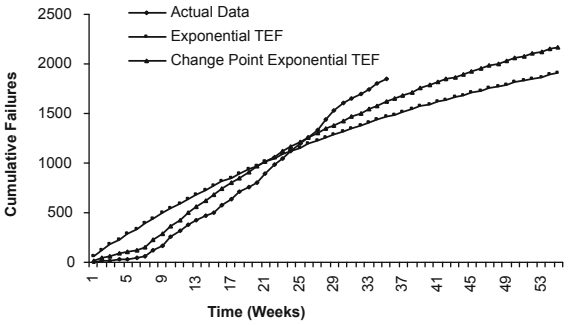


Fig. 5.14 Goodness of fit curve of exponential and modified exponential TEF

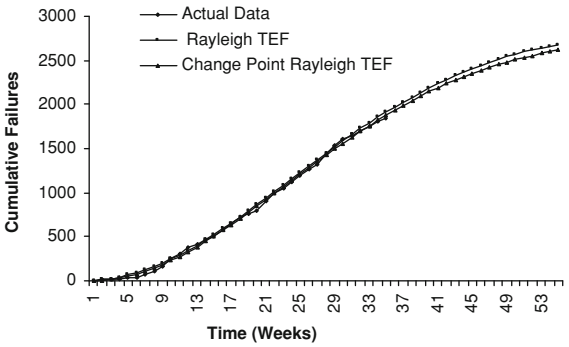


Fig. 5.15 Goodness of fit curve of Rayleigh and modified Rayleigh TEF

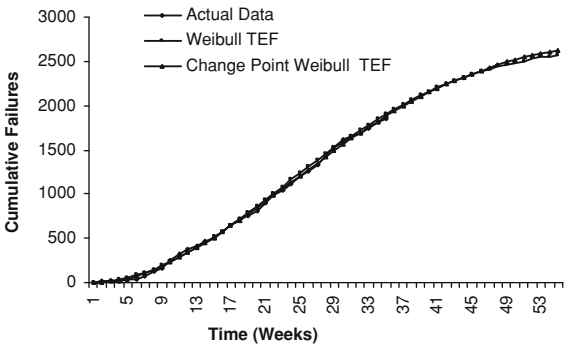


Fig. 5.16 Goodness of fit curve of Weibull and modified Weibull TEF

The modified Weibull test effort function best describes this data set. We estimated the test effort values corresponding to the observed data using the modified Weibull test effort function and using these values estimates of parameters of model M21 are obtained. For model M20 the actual test effort values are taken for the purpose of estimation. Estimated values of parameters of the SRGM are tabulated in Table 5.5. Goodness of fit curve for the SRGM M20 and M21 is shown in Fig. 5.17.

Table 5.5 Estimation results of models M20 and M21

Model	Estimated parameters							Comparison criteria	
	a	b, b_1	b_2	p, p_1	p_2	α, α_1	α_2	MSE	R^2
M20	1,393	0.0014	–	0.9927	–	0.01040	–	3,539.14	0.9893
M21	1,657	0.0012	0.0010	0.9254	0.9638	0.00005	0.00004	1,647.93	0.9930

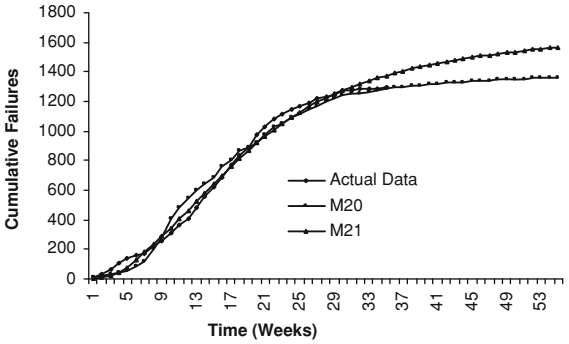


Fig. 5.17 Goodness of fit curve of models M20 and M21

The estimation results suggests that the model M21, i.e. testing efficiency and test effort based change point SRGM provides better fit as compared to the without change-point counterpart.

5.11.4 Application of Testing Effort Control Problem

Testing effort control problem discussed in Sect. 5.10 is widely used during the testing process to adjust the progress of testing. After certain period of continuation of the testing process, testing managers predict the status of the reliability level that can be achieved with the same level of testing after certain period of further testing. By the application of software release time problem managers are able to predict the delivery date of the software with a predetermined level of quality measure to be attained by that time. Sometimes the delivery date as determined from the optimization routine falls after the time when the software is scheduled to release. It is quite possible that in such a case many esteemed top management may not be ready to pay any sort of penalty cost due to late delivery, but they can spent extra resources now to accelerate the testing growth and achieve their required measure of quality by the scheduled delivery time. Now the problem is to determine how much resources are just sufficient to accelerate the testing so that the required quality measure can be attained by a specified time. Now we discuss a practical application of a testing effort control problem and illustrate how all these decisions can be made.

Failure Data Set

This data set is from Brooks and Motley [42]. The failure data set is for a radar system of size 124 KLOC (Kilo Lines of Code) tested for 35 weeks in which 1,301 faults were detected. In this data set a change-point is observed around the 17th observation, hence we assume $\tau = 17$th week.

The following SRGM is used in the application of testing effort control problem.

Model 22 (M22) Flexible/S-shaped test efforts based change-point SRGM [20]

$$m(t) = \begin{cases} a\left(\dfrac{1 - \mathrm{e}^{-b_1 W^*(t)}}{1 + \beta \mathrm{e}^{-b_1 W(t)}}\right) & 0 \le t \le \tau, \\ a\left[1 - \left(\dfrac{\left(1 + \beta \mathrm{e}^{-b_1 W(0)}\right)\left(1 + \beta \mathrm{e}^{-b_2 W(\tau)}\right)}{\left(1 + \beta \mathrm{e}^{-b_1 W(\tau)}\right)\left(1 + \beta \mathrm{e}^{-b_2 W(t)}\right)}\right)\mathrm{e}^{-(b_1 W^*(\tau) + b_2 W(t-\tau))}\right] & \tau < t \end{cases}$$

Consider the testing process at the end of 30th week. At this time moment the testing manager decides to release the software at the 35th week of testing with some desired level of reliability. First we truncate the data to 30 weeks and estimate the parameters of the SRGM developed due to Kapur et al. [20] (refer Sect. 5.10). To estimate the parameters of the SRGM first we estimate the parameter of the exponential, Rayleigh, Weibull and logistic testing effort functions and using these estimates, parameters of the SRGM are estimated. The estimation results of the testing effort functions are tabulated in Table 5.6. The Rayleigh curve fits best to these data. Parameters of the SRGM are estimated based on the Rayleigh effort function. The results of the estimation of SRGM are given in Table 5.7.

Using the results in Tables 5.6 and 5.7, we calculate the expected number of faults that can be removed with the same level of testing if testing is to be

Table 5.6 Estimation results for test effort functions

Test effort function	Estimated parameters			Comparison criteria	
	α	v	c, β	MSE	R^2
Exponential	962,053	0.000046	–	494.46	0.89
Rayleigh	3,254	0.000744	–	168.69	0.985
Weibull	3,767	0.000750	1.939	345.84	0.921
Logistic	1,895	0.172650	40.300	11,875.25	0.762

Table 5.7 Estimation results of model M22

Model	Estimated parameters				Comparison criteria	
	a	b_1	b_2	β	MSE	R^2
M22	1325	0.0028	0.0027	2.43	191.01	0.998

Table 5.8 Results of testing effort control problem

m^*	1,303	1,304	1,305	1,306	1,307
Faults to be removed between week (30, 35) of testing	36	37	38	39	40
Additional testing resources required	365.23	382.60	400.81	419.94	440.10

terminated by the end of 35th week. From Eq. (5.10.1) we get $m(T_2) = 1{,}302$ while $m(T_1) = 1{,}267$. It means that additional 35 faults (1,302 – 1,267) can be removed if we continue the testing for 5 weeks after 30th week. Now if want to release software after 35 weeks of testing and to attain a specified level of reliability we need to remove more than 35 faults in this time period and more resources should be added to accelerate the testing. Using Eqs. (5.10.5) and (5.10.6) we can determine the additional resources required. The estimated values of $W^{\#}$ with respect to different levels m^* are tabulated in Table 5.8.

Exercises

1. What are the two important techniques to handle the changes in processes mathematically? Give some important applications of change-point analysis.
2. Explain how change-point analysis is related to the measurement of software reliability growth during testing.
3. Describe the mathematics of change-point theory.
4. Develop an integrated flexible imperfect debugging model, if the fault removal intensity function is given as follows

$$\frac{\mathrm{d}m_r(t)}{\mathrm{d}t} = b(p,t)(a(t) - m_r(t))$$

with $b(p,t) = \begin{cases} b_1 p/1 + \beta \mathrm{e}^{-b_1 pt} & 0 \le t \le \tau, \\ b_2 p/1 + \beta \mathrm{e}^{-b_2 pt} & t > \tau \end{cases}$

and $a(t) = \begin{cases} a + \alpha_1 m_r(t) & 0 \le t \le \tau \\ a + \alpha_1 m_r(\tau) + \alpha_2 (m_r(t) - m_r(\tau)) & t > \tau \end{cases}$

5. Estimate the unknown parameters of the model developed in exercise 4 and the one discussed in Sect. 5.5.2 using the data of application in Sect. 5.11.1. Analyze and compare your results.
6. Incorporate the single change-point in the Yamada delayed S-shaped SRGM with respect to the test effort. The model without change-point is given as

$$m(t) = a(1 - (1 + bW(t))\mathrm{e}^{-bW(t)})$$

7. Fit the model developed in exercise 6 on data of application in Sect. 5.11.1. Reanalyze the results of this application to reflect the goodness of fit of this model.

References

1. Hackl TP, Westlund AH (1989) Statistical analysis of "structural change": an annotated bibliography. In: EMPEC, vol 14, pp 167–192
2. Khodadadi A, Asgharian M (2008) Change-point problem and regression: an annotated bibliography. COBRA Preprint Series. Article 44. http://biostats.bepress.com/cobra/ps/art44
3. Zhao M (1993) Change-point problems in software and hardware reliability. Commun Stat Theory Methods 22:757–768
4. Gupta A, Kapur R, Jha PC (2008) Considering testing efficiency in estimating software reliability based on testing variation dependent SRGM. Int J Reliab Qual Safety Eng 15(2):77–81
5. Chen J, Gupta AK (2001) On change-point detection and estimation. Commun Stat Simulation Comput 30(3):665–697
6. Hinkley DV (1970) Inference about the change-point in a sequence of binomial variable. Biometrika 57:477–488
7. Carlstein E (1988) Nonparametric change-point estimation. Ann Stat 16(1):188–197
8. Joseph L, Wolfson DB (1992) Estimation in multi-path change-point problems. Commun Stat Theory Methods 21(4):897–914
9. Xie M, Goh TN, Tang Y (2004) On changing points of mean residual life and failure rate function for some generalized Weibull distributions. Reliab Eng Syst Safety 84(3):293–299
10. Bae SJ, Kvam PH (2006) A change-point analysis for modeling incomplete burn in for light displays. IIE Trans 38(6):489–498
11. Zhao J, Wang J (2007) Testing the existence of change-point in NHPP software reliability models. Commun Stat Simulation Comput 36(3):607–619
12. Galeano P (2007) The use of cumulative sums for detection of change-points in the rate parameter of a Poisson process. Comput Stat Data Anal 51(12):6151–6165
13. Chang YP (1997) An analysis of software reliability with change-point models. NSC 85-2121-M031-003; National Science Council, Taiwan
14. Chang YP (2001) Estimation of parameters for non-homogeneous Poisson process: software reliability with change-point model. Commun Stat Simulation Comput 30(3):623–635
15. Shyur HJ (2003) A stochastic software reliability model with imperfect debugging and change-point. J Syst Softw 66:135–141
16. Zou FZ (2003) A change-point perspective on the software failure process. Softw Testing Verification Reliab 13:85–93
17. Huang CY (2005) Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Softw 76:181–194
18. Huang CY, Lin CT (2005) Reliability prediction and assessment of fielded software based on multiple change-point models. In: Proceedings 11th pacific rim international symposium on dependable computing (PRDC'05), pp 379–386
19. Lin CT, Huang CY (2008) Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. J Syst Softw 81:1025–1038
20. Kapur PK, Gupta A, Shatnawi O, Yadavalli VSS (2006) Testing effort control using flexible software reliability growth model with change-point. Int J Performability Eng special issue on Dependability of Software/Computing Systems 2:245–262
21. Kapur PK, Kumar A, Yadav K, Kumar J (2007) Software reliability growth modeling for errors of different severity using change-point. Int J Reliab Qual Safety Eng 14(4):311–326
22. Kapur PK, Singh VB, Anand S (2007) Effect of change-point on software reliability growth models using stochastic differential equation. In: 3rd International conference on reliability and safety engineering (INCRESE-2007), Udaipur, 7–19 Dec 2007, pp 320–333
23. Kapur PK, Kumar J, Kumar R (2008) A unified modeling framework incorporating change-point for measuring reliability growth during software testing. OPSEARCH J Oper Res Soc India 45(4):317–334

24. Kapur PK, Singh VB, Anand S, Yadavalli VSS (2008) Software reliability growth model with change-point and effort control using a power function of testing time. Int J Prod Res 46(3):771–787
25. Jelinski Z, Moranda P (1972) Software reliability research. In: Freiberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 465–484
26. Littlewood B (1981) Stochastic reliability growth: a model for fault removal in computer programs and hardware design. IEEE Trans Reliab R-30:312–320
27. Wagoner WL (1973) The final report on a software reliability measurement study. Aerospace Corporation, Report TOR-007 (4112), p 1
28. Schick GL, Wolverton RW (1973) Assessment of software reliability. In: Proceedings operations research. Physica-Verlag, Wurzburg Wein, pp 395–422
29. Inoue S, Yamada S (2008) Optimal Software release policy with change-point. In: IEEE international conference on industrial engineering and engineering management IEEM 2008, Singapore, 8–11 Dec 2008, pp 531–535
30. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R32(5):475–484
31. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
32. Sehgal VK, Kapur R, Yadav K, Kumar D (2010) Software reliability growth models incorporating change-point with imperfect fault removal and error generation. Int J Simulation Process Modeling (accepted for publication)
33. Zhang X, Pham H (2000) An analysis of factors affecting software reliability. J Syst Softw 50:43–56
34. Zhang X, Shin MY, Pham H (2001) Exploratory analysis of environmental factors for enhancing the software reliability assessment. J Syst Softw 57:73–78
35. Zhang XM, Jeske DR, Pham H (2002) Calibrating software reliability models when the test environment does not match the user environment. Appl Stochastic Models Business Industry 18:87–99
36. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R28(3):206–211
37. Pham H, Nordmann L, Zhang XA (1999) General imperfect-software-debugging model with S-shaped fault-detection rate. IEEE Trans Reliab 48(2):169–175
38. Liu HW, Yang XZ, Qu F, Dong J (2005) A software reliability growth model with bell-shaped fault detection rate function. Chin J Comput 28(5):908–913
39. Zhao J, Liu HW, Cui G, Yang XZ (2006) Software reliability growth model with change-point and environmental function. J Syst Softw 79:1578–1587
40. Ohba M (1984) Software reliability analysis models. IBM J Res Dev 28:428–443
41. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York. ISBN 0–07-044093-X
42. Brooks WD, Motley RW (1980) Analysis of discrete software reliability models—technical report (RADC-TR-80-84). Rome Air Development Center, New York
43. Yamada S, Hishitani J, Osaki S (1993) Software reliability growth model with Weibull testing effort: a model and application. IEEE Trans Reliab 42:100–105

Chapter 6
Unification of SRGM

6.1 Introduction

We are aware that it is the computer systems on which the entire modern information society rolls over. Computer hardware systems have attained high productivity, quality and reliability but it is still not true for the software systems. Software engineers and concerned managements put more labor for improving these characteristics of software nowadays. Unlike hardware components, every new software must be tested even though various techniques are employed throughout the software development process to satisfy software quality requirements. The achieved quality level through testing has no meaning unless it is measured quantitatively to build a confidence in the level of reliability achieved. Besides this, many decisions such as release time, those related to the postrelease can be made more accurately only if a quantitative measurement of quality is known. For example if we know the reliability level we can determine the post-delivery maintenance cost and warranty on the software more accurately and with more confidence. All this needs measurement technologies to assess the software quality. At this point we know that software reliability models based on stochastic and statistical principles are the important and most successful tools to assess software reliability quantitatively. Up to now we have discussed various software reliability modeling categories and several non-homogenous Poisson process (NHPP) based models under each category and will discuss more in the later chapters.

The existing NHPP models are formulated considering diverse testing and debugging (T&D) environments and have been applied successfully to typical reliability growth phenomenon observed during testing but not in general, i.e. a particular SRGM cannot be applied in general as the physical interpretation of the T&D is not general. A solution to this problem is to develop a unified modeling (UM) approach [1–9]. Although the SRGM existing in the literature considered one or other aspect of the software-testing but as mentioned above none can describe a general testing scenario. As such for any particular practical application

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_6,
© Springer-Verlag London Limited 2011

of reliability analysis one needs to study several models and then decide the most appropriate ones. The selected models are compared based on the results obtained and then a model is selected for further use. As an alternative following a unification approach several SRGM can be obtained from a single approach giving an insightful investigation on these models without making many distinctive assumptions. It can make our task of model selection and application much simpler than the usual methods. Establishment of unification methodology is one of the recent topics of research in software reliability modeling.

Some researchers have worked for formulating generalized approaches to robust software reliability assessment, which proved to be promising approachs for reliability estimation and prediction. The work in this area started with Shantikumar [1, 2] by proposing a generalized birth process model to describe Jelinski and Moranda [10] and GO model [11]. Langberg and Singpurwalla [3] showed that several Software Reliability Models (SRMs) can be comprehensively viewed from the Bayesian point of view. Miller [4] and Thompson Jr. [5] extended the Langberg and Singpurwalla's idea to a wider range of SRMs using the framework of theory of generalized order statistics (GOS) and record value. The NHPP models selection problem is reduced to a simple selection problem of fault-detection time distribution. Based on their result, the mean value function in NHPP models can be characterized by theoretical probability distribution function of fault-detection time. Gokhale et al. [6] and Gokhale and Trivedi [8] introduced the concept of testing coverage and proposed the similar modeling framework to the GOS theory. Chen and Singpurwalla [7] proved that all SRMs as well as the NHPP models developed in the past literature can be unified by self-exciting point processes. Huang et al. [9] discussed a unified scheme for discrete time NHPP models applying the concept of means. Xie et al. [23] proposed a unification scheme for modeling the fault detection and correction process. Kapur et al. [12, 13] proposed two approaches: one based on infinite server queuing theory and the other based on fault-detection time distribution. Using the UM approaches [12, 13, 23] several existing SRGM are obtainable, which have been developed considering diverse concepts of testing and debugging. In this chapter we will discuss these three unification methodologies and the SRGM obtained from them. In the later part of this chapter we have discussed the findings due to Kapur et al. [14] which yields the equivalence between the three approaches.

Before we proceed to the unification methodologies we feel it essential to discuss an important concept of software-testing. It is related to the existing time lag between fault-detection and its subsequent removal. In practice, fault removal is not immediate to its detection. Let us understand the software-testing process. It consists of exercising a program with intent to detect faults lying in it prior to software delivery to the users. This can be achieved by means of inspection, test runs and formal verification. On detection of a fault the debugging process starts. Fault debugging includes several intermediate steps such as failure report, fault isolation and correction with the subsequent verification. The goal of isolation activity is to identify the root cause of the fault, which is achieved through forming a hypothesis with the gathered information and then testing the hypothesis. As the fault is

isolated it is corrected and verified by the programmers. Fault correction personnel also formulate a hypothesis and make predictions based on the hypothesis. Furthermore, they run the software, observe its output and confirm the hypothesis on removal. The time to remove a fault depends on the complexity of the detected faults, the skills of the debugging team, the available manpower and the software development environment, etc. As such the assumption of immediate removal of fault on failure may not be realistic in many actual software-testing processes and provides a poor estimate of reliability. In practical testing scenario fault removal process follows detection process. This discussion enables us to know the importance of modeling the fault correction process (FCP) separately and simultaneously to the detection process. A few attempts have been made in the literature to model the fault-detection and correction process (FDCP) separately. Schneidewind [15] first modeled the FCP by using a constant delayed fault-detection process (FDP). Later, Xie and Zhao [16] extended the Schneidewind model using a time-dependent delay function. Yamada delayed S-shaped model [17] also considers the time lag between the fault correction and detection. Kapur and Younes [18] analyzed the software reliability considering the fault dependency and debugging time lag. Following the idea of using deterministic and random delay functions some researchers have emphasized importance of fault correction process modeling with FDP modeling [19–23]. The unified approaches we are going to discuss in this chapter incorporate the idea of modeling FDCP separately.

6.2 Unification Scheme for Fault Detection and Correction Process

When the information is available about both FDP and FCP, correction process has to be analyzed as a process separate from fault-detection. The analysis of fault correction mechanism follows similar to that for traditional NHPP-based SRGM. Each fault correction process is connected to a detection process as faults can only be corrected if they are detected. FCP can hence be assumed to be a delayed FDP. Therefore fault correction SRGM can be defined for all existing fault-detection SRGM by using the different forms of the time delay between these two processes. Following this approach and considering the importance of generalization and extension of the existing fault-detection SRGM recently Xie et al. [23] have proposed a unification scheme for modeling the FDCP. Using their scheme they obtained several SRGM for FDCP using distinct random delay functions. Wu et al. [22] further studied their unification scheme and obtained new SRGM using other type of random delay functions. This approach may be developed further in two ways: firstly, different existing NHPP models could be used to describe the FDP and using it we can obtain the SRGM for FCP; secondly, different time delay forms could be generated under different fault correction conditions.

6.2.1 Fault Detection NHPP Models

The mean value function, $m_d(t)$ of the FDP, $N(t)$ under the general assumptions of an NHPP-based SRGM with intensity $\lambda_d(t)$ satisfies

$$m_d(t) = \int_0^t \lambda_d(s)\,\mathrm{d}s \tag{6.2.1}$$

Using (6.2.1) we can obtain several existing fault-detection models, either concave or S-shaped. An exponential decreasing intensity describes a concave SRGM while an increasing then decreasing intensity describes an S-shaped FDP, which could be interpreted as a learning process.

6.2.2 Fault Correction NHPP Models

NHPP-based fault correction process is characterized by a mean value function $m_c(t)$ similar to $m_d(t)$. Mean value function of the FCP is obtainable from the $\lambda_d(t)$ using the delay function $\Delta(t)$ as

$$m_c(t) = \int_0^t \lambda_c(s)\,\mathrm{d}s = \int_0^t E[\lambda_d(s - \Delta(s))]\,\mathrm{d}s \tag{6.2.2}$$

Let us now discuss the various delay functions that can be used to describe the time lag between fault-detection and correction and obtain various existing SRGM for FDCP. It is assumed here that the mean value function of the GO model describes the detection process, i.e.

$$m_d(t) = a\left(1 - \mathrm{e}^{-bt}\right) \tag{6.2.3}$$

which implies

$$\lambda_d(t) = ab\mathrm{e}^{-bt} \tag{6.2.4}$$

Case 1: Constant Correction Time Assuming that each detected fault takes the same amount of time to be corrected, i.e. $\Delta(t) = \Delta$, with the known intensity for the FDP, the intensity function for the correction process is given as

$$\lambda_c(t) = \begin{cases} 0 & t < \Delta \\ \lambda_d(t - \Delta) & t \geq \Delta \end{cases} \tag{6.2.5}$$

With the intensity function (6.2.5) mean value function for the correction process is given as

$$m_c(t) = \begin{cases} 0 & t < \Delta \\ \int_{\Delta}^{t} \lambda_d(t-\Delta)\,\mathrm{d}t & t \geq \Delta \end{cases} \tag{6.2.6}$$

This can be further evaluated as

$$m_c(t) = m_d(t-\Delta) = a\left(1 - \mathrm{e}^{-b(t-\Delta)}\right); \quad t \geq \Delta \tag{6.2.7}$$

The model in (6.2.7) has the same form as Schneidewind's [15] correction model, although the latter is derived from the assumption that the fault correction rate is proportional to the number of faults to be corrected. Actually, these two assumptions are the same, as they both regard all of the detected faults as identical from the correction time point of view.

Case 2: Time-Dependent Correction Time Schneidewind's fault correction model was further studied by Xie and Zhao [16]. They proposed a time-dependent delay function to describe the time lag between failure observation and correction given as

$$\Delta(t) = \frac{\ln(1 + ct - c/b)}{b}; \quad c < b \tag{6.2.8}$$

Using this delay function the intensity function for the correction process is given as

$$\lambda_c(t) = \lambda_d(t - \Delta(t)) \tag{6.2.9}$$

Hence the mean value function for the correction process is given as

$$m_c(t) = \int_{\Delta}^{t} \lambda_d(t - \Delta(t))\,\mathrm{d}t = a\left(1 - (1 + ct)\mathrm{e}^{-bt}\right) \tag{6.2.10}$$

The model in (6.2.10) is the general form of the Yamada delayed S-shaped SRGM as in this model author has assumed $c = b$.

Case 3: Random Correction Time In cases 1 and 2 the correction time functions were deterministic. The deterministic assumptions on correction time are simplistic. In general it is not realistic as correction process is related to a human activity, which has lots of uncertainty in practice. Faults of any software are not usually same and their appearance sequence is random in system testing. Therefore, Xie et al. [23] proposed to model correction time with a random variable.

6.2.2.1 Exponentially Distributed Correction Time

Musa et al. [24] claimed that the correction time follows an exponential distribution. Following them Xie et al., defined the delay function as an exponential distribution, i.e.

$$\Delta(t) \sim \exp(\mu)$$

With an exponentially distributed delay function and a known intensity of the detection process the failure correction intensity function is given as

$$\lambda_c(t) = E[\lambda_d(t - \Delta(t))] = \int_0^\infty \lambda_d(t-x)\mu e^{-\mu x}\,dx \tag{6.2.11}$$

Hence the mean value function for the correction process with detection process described by GO model is given as

$$m_c(t) = \begin{cases} a\left(1-(1+bt)e^{-bt}\right)\mu = b \\ a\left(1-\frac{\mu}{\mu-b}e^{-bt}+\frac{b}{\mu-b}e^{-\mu t}\right)\mu \neq b \end{cases} \tag{6.2.12}$$

Further Xie et al. [23] and Wu et al. [22] proposed to use various other types of random distribution functions for modeling the correction time such as Gamma, Weibull, Erlang, Normal distribution, etc. Table 6.1 summarizes the various SRGM for the fault correction process obtained using distinct randomly distributed delay functions.

The unification scheme discussed above is a comprehensive study of the fault-detection and correction mechanism and offers a great flexibility for modeling

Table 6.1 Fault correction SRGM

Model	Probability density function of the correction time $\Delta(t)$	Mean value function of the SRGM for fault correction process ($m_c(t)$)
M1	Exponential time delay $g(x;\mu) = \mu e^{-\mu x}; \quad x \geq 0$	$a(1-(1+bt)e^{-bt}) \quad \mu = b$ $a\left(1-\frac{\mu}{\mu-b}e^{-bt}+\frac{b}{\mu-b}e^{-\mu t}\right) \quad \mu \neq b$
M2	Weibull time delay $g(x;\alpha,\beta) = \frac{\alpha}{\beta}\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x}{\beta}\right)^{\alpha}}; \quad x \geq 0$	$\sum_{i=0}^{\infty}\frac{(\beta b)^i}{i!}\int_0^t ab e^{-bt}\gamma\left(\frac{i}{\alpha}+1,t\right)dt$ where $\gamma(a,x)$ is incomplete Gamma function
M3	Erlang time delay $g(x;\alpha,\beta) = \frac{\beta^{\alpha}x^{\alpha-1}e^{-\beta x}}{(\alpha-1)!}; \quad x > 0$	$\frac{ab\beta^{\alpha}}{(\beta-b)^{k}(\alpha-1)!}\int_0^t e^{-bt}\gamma(\alpha,(\beta-b)t)\,dt$ where the shape parameter α is an integer, the rate parameter β is a real number and $\gamma(a,x)$ is the lower incomplete Gamma function
M4	Normal time delay $g(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\Pi}}e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$	$-ae^{(-bt+\mu b+(b\sigma)^2/2)}\begin{pmatrix}\Phi(t,b\sigma^2+\mu,\sigma)-\\+\Phi(0,b\sigma^2+\mu,\sigma)\end{pmatrix}$ $+a(\Phi(t,\mu,\sigma)-\Phi(0,\mu,\sigma))$
M5	Chi-square time delay $g(x;\beta) = \frac{(1/2)^{\beta/2}x^{(\beta/2-1)}e^{-x/2}}{\Gamma(\beta/2)}; \quad x \geq 0$	$\frac{ab\beta^{\alpha}}{(1-2b)^{\beta/2}\Gamma(\beta/2)}\int_0^t e^{-bt}\gamma\begin{pmatrix}\beta/2,\\(1-2b)t/2\end{pmatrix}dt$
M6	Gamma time delay $g(x;\alpha,\beta) = x^{\alpha-1}\frac{\beta^{\alpha}e^{-\beta x}}{\Gamma(\alpha)}; \quad x > 0$	$a\Gamma(t,\alpha,\beta) - \frac{ae^{-bt}}{(1-b\beta)^{\alpha}}\Gamma\left(t,\alpha,\frac{\beta}{(1-b\beta)}\right)$

the delay time function. With any known delay time function and based on an existing SRGM for FDP we can formulate the SRGM for fault correction process. Although the scheme analyzes the detection–correction mechanism very comprehensively, but it suffers from one major limitation. As mentioned above the deterministic assumptions on correction time are not realistic as correction process is related to a human activity and the faults manifested in any software are not usually same and their appearance sequence is random in system testing. As such it may not be appropriate to describe the time lag between their removals by same deterministic or randomly distributed delay function. One solution to this problem is to distinguish the faults based on their complexity of removal and accordingly forming the delay functions. The UM approach discussed in the next section incorporates this idea—a unified scheme based on the concept of infinite queues.

6.3 Unified Scheme Based on the Concept of Infinite Server Queue

Queuing models are very useful for several practical problems. These models have also been used successfully by the software engineering researchers for management and reliability estimation of software. Early attempts were related to the project staffing and software management [25, 26]. In the recent practices researchers have shown how to use queuing approaches to explain testing and debugging behavior of software. The underlying idea is that fault-detection phenomenon can be looked as an arrival in the queue and FCP can be seen as a service. If we assume the debugging activity starts as soon as a fault is detected, FDCP can be viewed as an infinite server queue. Inoue and Yamada [27] applied infinite server queuing theory to the basic assumptions of delayed S-shaped SRGM, i.e. FDP consists of successive failure detection and isolation processes considering time distribution of fault isolation process (FIP) and obtained several NHPP models describing FDP as a two-stage process.

The unification approach due to Inoue et al. describes FDP in two successive stages and no consideration is made for the removal phenomenon of the detected faults. Dohi et al. [28] proposed a unification method for NHPP models describing test input and program path searching times stochastically by an infinite server queuing theory. They assumed test cases are executed according to a homogeneous Poisson process (HPP) with parameter λ. Kapur et al. [12] proposed a comprehensive unified approach applying the infinite server queuing theory based on the basic assumptions of an SRGM defining the three level complexities of faults ([18], see section 2.4) with a consideration to FRP on the detection and isolation of a fault. The two separate approaches of modeling namely the time lag modeling between removal of fault on a failure observation and fault categorization incorporated in the unified scheme due to

Kapur et al. share a common thing that is the consideration of time lag between the failure observation, isolation and/or removal. It makes their methodology more general as it can be used to obtain several distinct categories of the models. These categories include models which consider testing a one-stage process with no fault categorization [11, 17] a two-stage process considering the various deterministic and random delay functions [22] and model which categorizes faults in two and three level complexity considering the time delay in failure observation, isolation and removal [18]. Let us now discuss their models in detail.

6.3.1 Model Development

Consider the case when a number of test cases are executed on software in accordance with an NHPP with rate $\lambda(t)$. The execution of test cases may result in the software failures. The failures observed at the end of execution of test cases form an arrival process. Here, number of failures observed at the end of testing period is equivalent to number of customers in the $M^*/G/\infty$ queuing system. Here arrival process, represented by M^* is an NHPP with mean $m(t)$ and service time has general distribution.

The Erlang model due to Kapur et al. [18] implies that a failure observation does not always imply that the fault is removed immediately. As in case of hard and complex faults the time spent in isolating and removing a fault on the observation of a failure is random due to the complexity of faults. In case of hard faults it is assumed that fault removal follows immediate to isolation while in case of complex fault delay happens in removal after the isolation. Hence first we need to discuss the concept of conditional distributions of arrival times (failures in this case) for developing the model based on infinite server queue theory [27].

6.3.1.1 Conditional Distribution of Arrival Times

Let $S_1, S_2, \ldots, S_n$ be the n arrival times of a counting process $\{N(t), t \geq 0\}$ which follows an NHPP with MVF $m(t)$ and the intensity function $\lambda(t)$. The conditional distribution of first arrival time, S_1, given that there was an event in the time interval $[0, t]$ [29], i.e. for $S_1 < t$, the conditional distribution is

$$\Pr\{S_1 \leq s_1 | N(t) = 1\} = \frac{m(s_1)}{m(t)} = \int_0^{s_1} \frac{\lambda(x)}{m(t)} \mathrm{d}x \tag{6.3.1}$$

Similarly, the joint conditional distribution of S_1 and S_2 is given as

$$\Pr\{S_1 \le s_1, S_2 \le s_2 | N(t) = 2\} = 2! \frac{m(s_1)(m(s_2) - m(s_1))}{m(t)^2}$$
$$= 2! \int_0^{s_1} \int_{s_1}^{s_2} \frac{\prod_{i=1}^{2} \lambda(x_i)}{m(t)^2} \mathrm{d}x_1 \, \mathrm{d}x_2 \qquad (6.3.2)$$

where $s_1 < s_2 \le t$. Hence if $N(T) = n$, the joint conditional distribution of n arrival times is given by

$$\Pr\{S_1 \le s_1, S_2 \le s_2, \ldots, S_n \le s_n | N(t) = n\} = n! \int_0^{s_1} \int_{s_1}^{s_2} \cdots \int_{s_{n-1}}^{s_n} \frac{\prod_{i=1}^{n} m(x_i)}{m(t)^n} \mathrm{d}x_1 \, \mathrm{d}x_2 \ldots \mathrm{d}x_n \qquad (6.3.3)$$

Therefore, the joint conditional distribution of n arrival times given that $N(T) = n$ is given as

$$\Pr\{t_1, t_2, \ldots, t_n | N(t) = n\} = n! \frac{\prod_{i=1}^{n} \lambda(t_i)}{m(t)^n} \qquad (6.3.4)$$

Equation (6.3.4) implies that if $N(T) = n$ the unordered random variables of n arrival times S_1, S_2, …, S_n are independent and identically distributed with the density

$$f(x) = \begin{cases} \dfrac{\lambda(x)}{m(t)} & 0 \le x \le t \\ 0 & \text{otherwise} \end{cases} \qquad (6.3.5)$$

Also if $s < t$ and $0 \le m \le n$, then

$$\Pr\{N(t) = m | N(t) = n\} = \frac{\Pr\{N(t-s) = n - m, N(s) = m\}}{\Pr\{N(t) = n\}}$$
$$= \binom{n}{m} \left(\frac{m(s)}{m(t)}\right)^m \left(1 - \frac{m(s)}{m(t)}\right)^{n-m} \qquad (6.3.6)$$

Equation (6.3.6) means that the conditional distribution of $N(s)$ given $N(T) = n$ follows a binomial distribution with parameter $(n, (m(s)/m(t)))$.

6.3.2 *Infinite Server Queuing Model*

The model is based on the following assumptions:

1. Faults in the software system are classified as complex, hard and simple.
2. Time delay between the failure observation and its subsequent removal represents the complexity of faults.

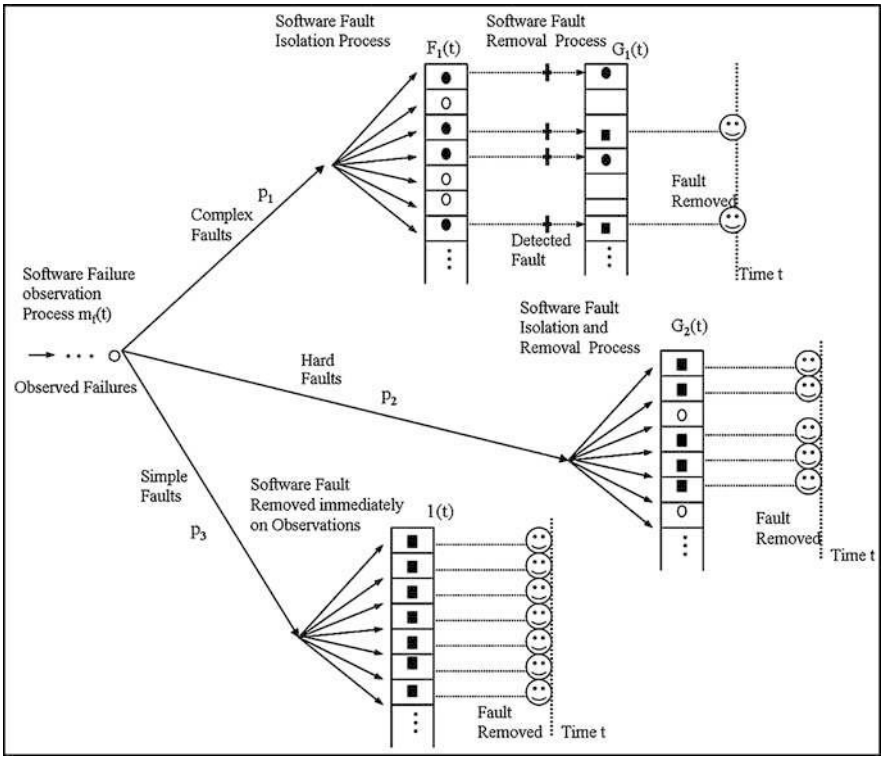


Fig. 6.1 Physical interpretation of the infinite server queuing model

3. The expected cumulative number of software failures is observed according to an NHPP with the MVF $m_f(t)(m_{fi}(t))$ for type i fault ($i = 1, 2, 3$) and the intensity function $\lambda(t)(\lambda_i(t))$ for type i fault ($i = 1, 2, 3$).

The model can be easily explained with Fig. 6.1.

6.3.2.1 Model for Complex Faults

Assumptions for fault isolation and removal process of complex faults are

1. For complex faults the observed software failures are analyzed in the FIP, which results in the detection of faults corresponding to observed failures.
2. The fault isolation times are assumed to be independent with a common distribution $F_1(t)$.
3. Fault removal process follows the detection process in which the detected complex faults are removed.
4. The fault removal times are assumed to be independent with a common distribution $G_1(t)$.

Let the counting processes $\{X_1(t), t \geq 0\}, \{R_1(t), t \geq 0\}, \{N_1(t), t \geq 0\}$ represent the cumulative number of software failures observed, faults isolated and faults removed, respectively, up to time t corresponding to the complex faults and the test begun at time $t = 0$. Then the distribution of $N_1(t)$ is given by

$$\Pr\{N_1(t) = n\} = \sum_{j=0}^{\infty} \Pr\{N_1(t) = n | X_1(t) = j\} \frac{\left(m_{f1}(t)\right)^j \mathrm{e}^{-m_{f1}(t)}}{j!} \tag{6.3.7}$$

If failure observations count is j then probability that n faults are removed via the fault isolation and removal process is given as

$$\Pr\{N_1(t) = n | X_1(t) = j\} = \binom{j}{n} (p_1(t))^n (1 - p_1(t))^{j-n} \tag{6.3.8}$$

where $p_1(t)$ is the probability that an arbitrary fault is removed by time t, which can be defined using the Stieltjes convolution and the concept of the conditional distribution of arrival times, given as

$$p_1(t) = \int_0^t \int_0^{t-u} G_1(t - u - v)\, \mathrm{d}F_1(u) \frac{\mathrm{d}m_{f1}(v)}{m_{f1}(t)} \tag{6.3.9}$$

The distribution function of cumulative number of faults removed up to time t using Eqs. (6.3.8) and (6.3.9) is given as a

$$\begin{aligned} \Pr\{N_1(t) = n\} = & \left(\int_0^t \int_0^{t-u} G_1(t - u - v)\, \mathrm{d}F_1(u)\, \mathrm{d}m_{f1}(v) \right)^n \\ & \times \frac{\mathrm{e}^{-\int_0^t \int_0^{t-u} G_1(t-u-v)\, \mathrm{d}F_1(u)\, \mathrm{d}m_{f1}(v)}}{n!} \end{aligned} \tag{6.3.10}$$

Equation (6.3.10) describes that $N_1(t)$ follows an NHPP with MVF $\int_0^t \int_0^{t-u} G_1(t - u - v)\, \mathrm{d}F_1(u)\, \mathrm{d}m_{f1}(v)$ i.e.

$$m_1(t) = \int_0^t \int_0^{t-u} G_1(t - u - v)\, \mathrm{d}F_1(u)\, \mathrm{d}m_{f1}(v) \tag{6.3.11}$$

Hence knowing the MVF $m_{f1}(\cdot)$ and distributions of $F_1(\cdot)$ and $G_1(\cdot)$ one can compute the MVF of a three-stage fault-detection and removal process for the various existing SRGM.

6.3.2.2 Model for Hard Faults

Assumptions for fault isolation/removal process of hard faults are

1. For hard faults the observed software failures are analyzed in the FIP which result in the detection of faults corresponding to observed failures, the detected faults can be removed immediately with no delay (i.e. the removal time distribution is a unit function).
2. The fault isolation and removal times are assumed to be independent with a common distribution $G_2(t)$.

Let the counting processes $\{X_2(t), t \geq 0\}, \{N_2(t), t \geq 0\}$ represent the cumulative number of software failures observed and faults detected/removed, respectively, up to time t corresponding to the hard faults and the test begun at time $t = 0$. Then the distribution of $N_2(t)$ is given by

$$\Pr\{N_2(t) = n\} = \sum_{j=0}^{\infty} \Pr\{N_2(t) = n | X_2(t) = j\} \frac{(m_{f2}(t))^j \mathrm{e}^{-m_{f2}(t)}}{j!} \tag{6.3.12}$$

If failure observations count is j then probability that n faults are removed via the fault isolation/removal process is given as

$$\Pr\{N_2(t) = n | X_2(t) = j\} = \binom{j}{n} (p_2(t))^n (1 - p_2(t))^{j-n} \tag{6.3.13}$$

where $p_2(t)$ is the probability that an arbitrary fault is removed by time t, which can be defined using the Stieltjes convolution and the concept of the conditional distribution of arrival times, given as

$$p_2(t) = \int_0^t G_2(t-u) \frac{\mathrm{d}m_{f2}(u)}{m_{f2}(t)} \tag{6.3.14}$$

The distribution function of cumulative number of faults removed up to time t using Eqs. (6.3.13) and (6.3.14) is given as

$$\Pr\{N_2(t) = n\} = \left(\int_0^t G_2(t-u)\, \mathrm{d}m_{f2}(u) \right)^n \frac{\mathrm{e}^{-\int_0^t G_2(t-u)\, \mathrm{d}m_{f2}(u)}}{n!} \tag{6.3.15}$$

Equation (6.3.15) describes that $N_2(t)$ follows an NHPP with MVF $\int_0^t G_2(t-u)\, \mathrm{d}m_{f2}(u)$ i.e.

$$m_2(t) = \int_0^t G_2(t-u)\, \mathrm{d}m_{f2}(u) \tag{6.3.16}$$

Hence knowing the MVF $m_{f2}(\cdot)$ and distribution of $G_2(\cdot)$ we can compute the MVF of a two-stage fault-detection and removal process for the various existing SRGM.

6.3.2.3 Model for Simple Faults

For simple faults it is assumed that the faults are removed as soon as they are observed and hence if $\{X_3(t), t \geq 0\}, \{N_3(t), t \geq 0\}$ are the counting processes represent the cumulative number of software failures observed and removed, respectively, up to time t corresponding to the simple faults and the test begun at time $t = 0$ then the distribution of $N_3(t)$ is given by

$$\Pr\{N_3(t) = n\} = \sum_{j=0}^{\infty} \Pr\{N_3(t) = n | X_3(t) = j\} \frac{\left(m_{f3}(t)\right)^j \mathrm{e}^{-m_{f3}(t)}}{j!} \tag{6.3.17}$$

If failure observations count is j then probability that n faults are removed via the fault isolation/removal process is given as

$$\Pr\{N_3(t) = n | X_3(t) = j\} = \binom{j}{n} (p_3(t))^n (1 - p_3(t))^{j-n} \tag{6.3.18}$$

where $p_3(t)$ is the probability that an arbitrary fault is removed by time t, which can be defined using the Stieltjes convolution and the concept of the conditional distribution of arrival times, given as

$$p_3(t) = \int_0^t G_3(t-u) \frac{\mathrm{d}m_{f3}(u)}{m_{f3}(t)} = \int_0^t 1(t-u) \frac{\mathrm{d}m_{f3}(u)}{m_{f3}(t)} \tag{6.3.19}$$

The distribution function of cumulative number of faults removed up to time t using Eqs. (6.3.18) and (6.3.19) is given as

$$\begin{aligned} \Pr\{N_3(t) = n\} &= \left(\int_0^t 1(t-u)\, \mathrm{d}m_{f3}(u) \right)^n \frac{\mathrm{e}^{-\int_0^t 1(t-u)\, \mathrm{d}m_{f3}(u)}}{n!} \\ &= \frac{\left(m_{f3}(t)\right)^n \mathrm{e}^{-m_{f3}(t)}}{n!} \end{aligned} \tag{6.3.20}$$

Equation (6.3.20) describes that $N_3(t)$ follows an NHPP with MVF $m_{f3}(t)$. Hence,

$$m_3(t) = \int_0^t 1(t-u)\, \mathrm{d}m_{f3}(u) = m_{f3}(t) \tag{6.3.21}$$

6.3.2.4 Model for Total FRP

If $\{N(t),\ t \geq 0\}$ are the counting processes that represent the cumulative number of software fault removals up to time t then $N(t)$ is derived as

$$\begin{aligned} N(t) &= N_1(t) + N_2(t) + N_3(t) \\ &= \frac{(m_1(t) + m_2(t) + m_3(t))^n}{n!} \mathrm{e}^{-[m_1(t)+m_2(t)+m_3(t)]} \end{aligned} \tag{6.3.22}$$

hence

$$m(t) = m_1(t) + m_2(t) + m_3(t) \tag{6.3.23}$$

where $m_1(t)$, $m_2(t)$ and $m_3(t)$ are given by (6.3.11), (6.3.16) and (6.3.23). It may be noted here in view of simplifying the computations of $m_1(t)$, $m_2(t)$ and $m_3(t)$ we can use the associative property of Stieltjes convolutions and can be rewritten as

$$\begin{aligned} m_1(t) &= \int_0^t \int_0^{t-u} G_1(t-u-v)\, \mathrm{d}F_1(u)\, \mathrm{d}m_{f1}(v) \\ &= \int_0^t \int_0^{t-u} m_{f1}(t-u-v)\, \mathrm{d}F_1(u)\, \mathrm{d}G_1(v) \end{aligned} \tag{6.3.24}$$

$$m_2(t) = \int_0^t G_2(t-u)\, \mathrm{d}m_{f2}(u) = \int_0^t m_{f2}(t-u)\, \mathrm{d}G_2(u) \tag{6.3.25}$$

$$m_3(t) = \int_0^t 1(t-u)\, \mathrm{d}m_{f3}(u) = m_{f3}(t) = \int_0^t m_{f3}(t-u)\, \mathrm{d}1(u) \tag{6.3.26}$$

Using the above modeling approach we can explain a number of existing SRGM formulated for different T&D scenario. In the next section we will discuss how this unified approach can be used to obtain MVF of the various existing SRGM.

6.3.3 *Computing Existing SRGM for the Unified Model Based on Infinite Queues*

The unified model given by Eqs. (6.3.23) to (6.3.26) characterizes the time-dependent behavior of fault-detection and removal phenomenon by determining $m_{fi}(t)$, $i = 1, 2, 3$, $F_1(t)$ and $G_i(t)$, $i = 1, 2$ and many existing SRGM formulated under different testing scenarios can be obtained. Accordingly, we can easily reflect the phenomenon of successive software failure occurrence, fault-isolation

and fault removal depending on the testing scenario and assumptions on the fault type/debugging process. Hence this modeling approach can be considered as a general description of several existing NHPP models.

If we assume

$$m_{fi}(t) = a_i\left(1 - e^{-b_i t}\right), \quad i = 1, 2, 3$$

$$F_1(t) = 1 - e^{-b_1 t}$$

and

$$G_i(t) = 1 - e^{-b_i t}, \quad i = 1, 2$$

then $m_i(t)$, $i = 1, 2, 3$ describe the MVF of FRP for complex, hard and simple faults and using (6.3.23) we obtain the MVF of total FRP for the Kapur et al. [18] Generalized Erlang model.

Similarly, we can describe the various other existing SRGM from our UM approach. We can obtain the models, which consider the various levels (1, 2, 3) of fault complexity, as well as several models, which consider each software fault to be of same type. It is very important to note here that this unification scheme can be used to obtain all models obtained with the unification scheme discussed in Sect. 6.2, which considers the fault-detection and the time delay in fault observation. All types of correction models with constant, time-dependent as well as random time delay function are obtainable from this scheme. The authors have obtained various fault-detection correction models from this scheme.

Table 6.2 summarizes the relationships between unified infinite server modeling approach and some of the existing NHPP-based SRGM.

The table summarizes models obtained from UM approach based on the concept of infinite queues. Many other existing models can also be obtained similarly.

6.3.4 A Note on Random Correction Times

Kapur et al. [14] have also explained the particular situations, when different random delay functions viz. Exponential, Weibull, Gamma, etc. are useful and need—to be considered. Here we discuss the particular use of some randomly distributed delay functions or correction times proposed by Xie et al. [23].

6.3.4.1 Exponential Distribution for Removal Times

This is the most simple and widely used distribution in reliability engineering modeling because it has a constant rate. It indicates the uniform distribution of faults in the software code where each and every fault has same probability for its removal. The *pdf* for exponential distribution is given by

Table 6.2 Some existing SRGM obtained using unified modeling approach based on infinite queues

Model	$m_{fi}(t)\,(m_f(t))$	$F_1(t)$	$G_1(t)$	$G_2(t)$ or $\begin{pmatrix} g_2(t) \\ \text{the pdf of} \\ G_2(t) \end{pmatrix}$	$m(t)$	Comments
M1	$ap_i\left(1-\mathrm{e}^{-b_i t}\right)$ $i=1,2,3$	$1(t)$	$1(t)$	$1(t)$	$ap_1\left(1-\mathrm{e}^{-b_1 t}\right)$ $+ ap_2\left(1-\mathrm{e}^{-b_2 t}\right)$ $+ ap_3\left(1-\mathrm{e}^{-b_3 t}\right)$	Assumes three types of faults and each MVF of FRP is described by GO model with different FRR
M2	$ap_i\left(1-\mathrm{e}^{-b_i t}\right)$ $i=1,2,3$	$T\sim\exp(b_1)$	$T\sim\exp(b_1)$	$T\sim\exp(b_2)$	$ap_1\left(1-\left(1+b_1 t+\frac{b_1^2 t^2}{2}\right)\mathrm{e}^{-b_1 t}\right)$ $+ ap_2\left(1-(1+b_2 t)\mathrm{e}^{-b_2 t}\right)$ $+ ap_3\left(1-\mathrm{e}^{-b_3 t}\right)$	Kapur et al. generalized Erlang SRGM with different FRR for each type of fault
M3	$ap_i\left(1-\mathrm{e}^{-bt}\right)$ $i=1,2,3$	$T\sim\exp(b)$	$T\sim\exp(b)$	$T\sim\exp(b)$	$ap_1\left(1-\left(1+bt+\frac{b^2 t^2}{2}\right)\mathrm{e}^{-bt}\right)$ $+ ap_2\left(1-(1+bt)\mathrm{e}^{-bt}\right)$ $+ ap_3\left(1-\mathrm{e}^{-bt}\right)$	Kapur et al. generalized Erlang SRGM with same FRR for each type of fault

(continued)

Table 6.2 (continued)

Model	$m_{fi}(t)\left(m_f(t)\right)$	$F_1(t)$	$G_1(t)$	$G_2(t)$ or $\begin{pmatrix} g_2(t) \\ \text{the pdf of} \\ G_2(t) \end{pmatrix}$	$m(t)$	Comments
M4	$ap_i\left(1-\mathrm{e}^{-b_i t}\right)$ $i=1,2,3$	$T\sim\exp(b_4)$	$T\sim\exp(b_5)$	$T\sim\exp(b_6)$	$ap_1\left(1-\frac{R}{S}\right)$ $+ap_2\left(1-\frac{1}{(b_2-b_6)}\begin{pmatrix} b_2\mathrm{e}^{-b_6 t} \\ -b_6\mathrm{e}^{-b_2 t}\end{pmatrix}\right)$ $+ap_3\left(1-\mathrm{e}^{-b_3 t}\right)$ $R=\begin{pmatrix} b_1^2\begin{pmatrix} b_4\mathrm{e}^{-b_5 t} \\ -b_5\mathrm{e}^{-b_4 t}\end{pmatrix}+ \\ b_4^2\begin{pmatrix} b_5\mathrm{e}^{-b_1 t} \\ -b_1\mathrm{e}^{-b_5 t}\end{pmatrix}+ \\ b_5^2\begin{pmatrix} b_1\mathrm{e}^{-b_4 t}- \\ b_4\mathrm{e}^{-b_1 t}\end{pmatrix}\end{pmatrix}$ $S=\begin{pmatrix}(b_1-b_4)\\(b_4-b_5)\\(b_5-b_1)\end{pmatrix}$	Generalized Erlang SRGM with different rates of failure observation, fault isolation and/or removal for each fault complexity

(continued)

Table 6.2 (continued)

Model	$m_{fi}(t)\big(m_f(t)\big)$	$F_1(t)$	$G_1(t)$	$G_2(t)$ or $\begin{pmatrix} g_2(t) \\ \text{the pdf of} \\ G_2(t) \end{pmatrix}$	$m(t)$	Comments
Models with one type of fault						
M5	$a\big(1-\mathrm{e}^{-b_1 t}\big)$	$T\sim\exp(b_2)$	$T\sim\exp(b_3)$	–	$a\big(1-\frac{R}{S}\big)$ $R=\begin{pmatrix} b_1^2\big(b_2\mathrm{e}^{-b_3 t}-b_3\mathrm{e}^{-b_2 t}\big)+ \\ b_2^2\big(b_3\mathrm{e}^{-b_1 t}-b_1\mathrm{e}^{-b_3 t}\big)+ \\ b_3^2\big(b_1\mathrm{e}^{-b_2 t}-b_2\mathrm{e}^{-b_1 t}\big) \end{pmatrix}$ $S=\begin{pmatrix} (b_1-b_2) \\ (b_2-b_3) \\ (b_3-b_1) \end{pmatrix}$	K-3 stage model with different rates
M6	$a\big(1-\mathrm{e}^{-bt}\big)$	–	–	$g(x;b)$ $=b\mathrm{e}^{-bx}$ $x>0$	$a\big(1-(1+bt)\mathrm{e}^{-bt}\big)$	Exponential distributed removal time delay model
M7	$a\big(1-\mathrm{e}^{-bt}\big)$	–	–	$g(x;\mu)$ $=\mu\mathrm{e}^{-\mu x}$ $x>0$	$a\begin{pmatrix} 1-\frac{\mu}{\mu-b}\mathrm{e}^{-bt} \\ +\frac{b}{\mu-b}\mathrm{e}^{-\mu t} \end{pmatrix}$	Exponential distributed removal time delay model with different parameters
M8	$a\big(1-\mathrm{e}^{-bt}\big)$	–	–	$g(x;\mu,\sigma)$ $=\frac{1}{\sigma\sqrt{2\Pi}}S$ $S=\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$ $\Phi((0,t),\mu,\sigma)=\int_0^t g(x;\mu,\sigma)$	$a\begin{pmatrix} \begin{pmatrix} -\mathrm{e}^{\left(-bt+\mu b+(b\sigma)^2/2\right)} \\ \Phi\big((0,t),b\sigma^2+\mu,\sigma\big) \end{pmatrix} \\ +\Phi((0,t),\mu,\sigma) \end{pmatrix}$	Normal distributed removal time delay model

(continued)

Table 6.2 (continued)

Model	$m_{fi}(t)\left(m_f(t)\right)$	$F_1(t)$	$G_1(t)$	$G_2(t)$ or $\begin{pmatrix} g_2(t) \\ \text{the pdf of} \\ G_2(t) \end{pmatrix}$	$m(t)$	Comments
M9	$a\left(1-\mathrm{e}^{-bt}\right)$	–	–	$g(x;\alpha,\beta) = \dfrac{\begin{pmatrix} x^{\alpha-1}\beta^{\alpha} \\ \mathrm{e}^{-\beta x} \end{pmatrix}}{\Gamma(\alpha)}$, $x>0$ $\Gamma(t,\alpha,\beta) = \int_0^t g(x;\alpha,\beta)$	$a(\Gamma(t,\alpha,\beta)-S)$ $S = \begin{pmatrix} \dfrac{\mathrm{e}^{-\mathrm{bt}}}{(1-b\beta)^{\alpha}} \\ \Gamma\left(t,\alpha,\dfrac{\beta}{(1-b\beta)}\right) \end{pmatrix}$	Gamma distributed removal time delay model
M10	at	–	–	$g(x;\alpha,\beta) = \begin{pmatrix} \alpha m(\alpha x)^{m-1} \\ \mathrm{e}^{-(\alpha x)^m} \end{pmatrix}$, $x>0$ Γ_1 is Gamma function	$ap_2\left(t-\frac{S}{m\alpha}\right)$ $S = m\Gamma_1\left(1+\dfrac{1}{m}\right) - \Gamma_2\left(\dfrac{1}{m},(\alpha t)^m\right)$ Γ_2 is upper incomplete Gamma function	Weibull distributed removal time delay model

(continued)

Table 6.2 (continued)

Model	$m_{fi}(t)\big(m_f(t)\big)$	$F_1(t)$	$G_1(t)$	$G_2(t)$ or $\begin{pmatrix} g_2(t) \\ \text{the pdf of} \\ G_2(t) \end{pmatrix}$	$m(t)$	Comments
Some new models for three level categorization of faults						
M11	$ap_i\big(1-\mathrm{e}^{-b_i t}\big)$ $i=1,2,3$	$1(t)$	$g(x;\alpha,\beta)=$ $x^{\alpha-1}\dfrac{\beta^{\alpha}\mathrm{e}^{-\beta x}}{\Gamma(\alpha)}$, $x>0$ $\Gamma(t,\alpha,\beta)=\int_0^t g(x;\alpha,\beta)$	$T\sim\exp(b_2)$	$ap_1(\Gamma(t,\alpha,\beta)-S)$ $+ap_2\big(1-(1+b_2t)\mathrm{e}^{-b_2t}\big)$ $+ap_3\big(1-\mathrm{e}^{-b_3t}\big)$ $S=\begin{pmatrix}\dfrac{\mathrm{e}^{-b_1 t}}{(1-b_1\beta)^{\alpha}} \\ \Gamma\left(t,\alpha,\dfrac{\beta}{(1-b_1\beta)}\right)\end{pmatrix}$	Three types of faults and FRP is described by GO model for simple YDSM for hard and Gamma time delay model for complex faults
M12	$ap_i\big(1-\mathrm{e}^{-b_i t}\big)$ $i=1,2,3$	$1(t)$	$g(x;\mu,\sigma)=\dfrac{1}{\sigma\sqrt{2\Pi}}S$ $S=\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$ $\Phi((0,t),\mu,\sigma)=\int_0^t g(x;\mu,\sigma)$	$T\sim\exp(b_2)$	$ap_1\begin{pmatrix}\left(-\mathrm{e}^{\left(-b_1t+\mu b_1+\frac{(b_1\sigma)^2}{2}\right)}\right) \\ \Phi((0,t),b_1\sigma^2+\mu,\sigma) \\ +\Phi((0,t),\mu,\sigma)\end{pmatrix}$ $+ap_2\big(1-(1+b_2t)\mathrm{e}^{-b_2t}\big)$ $+ap_3\big(1-\mathrm{e}^{-b_3t}\big)$	Three types of faults and FRP is described by GO model for simple, YDSM for hard and normal time delay model for complex faults

$$g(x;b) = be^{-bx} \tag{6.3.27}$$

Here b is the parameter of exponential distribution and it represents the mean rate at which the observed/isolated faults are removed. Here removals are assumed to take place at a constant rate.

Though in most of the software-testing projects, for sake of simplicity, the removal times are assumed to follow exponential distribution, but to achieve a more flexible modeling of removal times, we can use Weibull or Gamma distribution. Both of these distributions are generalization of Exponential distribution only and have very similar shapes.

6.3.4.2 Weibull Distribution for Removal Times

It can represent different types of curves depending on the values of its shape parameter. It is very appropriate for representing the processes with fluctuating rate i.e. increasing/decreasing rates. The *pdf* for Weibull distribution is given by

$$g(x;\alpha,\beta) = \alpha\beta(\alpha x)^{\beta-1}e^{-(\alpha x)^{\beta}}, \quad x > 0 \tag{6.3.28}$$

Here α, β are the parameters of Weibull distribution where β is shape parameter and α is scale parameter. When the shape parameter $0 < \beta < 1$, the removal rate decreases monotonically over time, for $\beta = 1$, the removal rate is constant and for $\beta > 1$, the removal rate increases monotonically over time. For $\beta = 1$, it reduces to exponential distribution, for $\beta = 2$ it is same as Rayleigh distribution and for $\beta = 3.4$ it behaves like Normal distribution.

6.3.4.3 Gamma Distribution for Removal Times

Gamma distribution is an extension of exponential distribution where the fault removal consists of multiple steps, e.g., generation of failure report, its analysis and correction time followed by verification and validation. The *pdf* for Gamma distribution is given by:

$$g(x;\alpha,\beta) = x^{\alpha-1}\frac{\beta^{\alpha}e^{-\beta x}}{\Gamma(\alpha)}, \quad x > 0 \tag{6.3.29}$$

Here α, β are the shape and scale parameters of Gamma distribution and represent the distribution of α number of independently and identically distributed exponential random variables, each with parameter β. This property of gamma distribution makes it appropriate for modeling processes consisting of a number of steps.

6.3.4.4 Normal Distribution for Removal Times

During testing, there are numerous factors, which affect the fault correction process. These factors can be internal, e.g., defect density, complexity of the faults, the internal structure of the software or the factors can be external and come from the testing environment itself, e.g., design of the test cases, skill of the testers/test case designers, testing effort availability/consumption, etc. This two-parameter distribution can describe the correction times quite well for the cases where correction time depends on multiple factors. The *pdf* for Normal distribution is given by:

$$g(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\Pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{6.3.30}$$

Here μ, σ are the location and scale parameters of Normal distribution. They represent mean and standard deviation of Normal distribution, respectively.

6.4 A Unified Approach for Testing Efficiency Based Software Reliability Modeling

The unified approaches discussed yet can be used to obtain SRGM developed under perfect debugging testing profile. As highlighted in Chap. 3 incorporation of effect of testing efficiency is very important while developing an SRGM. Inclusion of testing efficiency considerations in SRGM enables us to compute a more appropriate estimate of reliability growth during both testing and operational phases of SDLC. After the detection of a fault during the removal a fault may not be perfectly repaired or a new fault can be generated. In the first case we again come across a failure due to a fault, which has already been detected, resulting in more number of failures than removals. While in the second case more faults are observed compared to the initial number estimated in the infinite test period. However both of these cases make the testing and debugging environment entirely different from the perfect debugging environment. Hence the physical structure of software reliability modeling under imperfect debugging environment is different from software reliability modeling under perfect debugging. However it is very important to know here that an imperfect debugging model corresponds to a perfect debugging model when the estimated values of the parameters of testing efficiency attain an insignificant value, i.e. the case of perfect repairs and no generation applied to an SRGM developed under imperfect debugging environment. For the literature of SRGM developed incorporating imperfect debugging environment refer to Chap. 3.

In this section we discuss a unification scheme of SRGM, which can be used to obtain almost all of the SRGM developed under imperfect debugging environment

in the literature up to now. The said unification scheme can also be used to formulate many other SRGM under imperfect debugging environment as it is capable of handling any general distribution function and is thus an important step toward the unification of the NHPP software reliability measurement models, which rely on specific distribution functions.

The unified scheme of SRGM proposed due to Kapur et al. [13] discussed here is an insightful investigation for the study of general models without making many assumptions. They proposed two types of schemes for generalized imperfect non-homogeneous Poisson process (GINHPP) software reliability models, when there is no differentiation between failure observation/detection and fault removal/correction processes i.e. a fault is removed as soon as it is detected/observed (GINHPP-1). Second, when we incorporate the time delay between the fault observation and correction processes (GINHPP-2).

6.4.1 *Generalized SRGM Considering Immediate Removal of Faults on Failure Observation Under Imperfect Debugging Environment*

Under the general assumptions (see Chap. 2) of NHPP-based software reliability model under perfect debugging environment the mean value function of the generalized SRGM can be represented as [6, 8, 24]

$$m(t) = aF(t) \tag{6.4.1}$$

where a is the finite number of faults detected in the infinite testing time, $F(t)$ is a distribution function.

Hence the instantaneous failure intensity $\lambda(t)$ is given as

$$\lambda(t) = aF^{'}(t) \tag{6.4.2}$$

The above equation can be rewritten as

$$\lambda(t) = (a - m(t))\frac{F^{'}(t)}{1 - F(t)} = (a - m(t))s(t) \tag{6.4.3}$$

where $s(t)$ is the failure occurrence/observation/detection rate per remaining fault of the software, or the rate at which the individual faults manifest themselves as failures during testing or hazard rate. The expression $[a - m(t)]$ denotes the expected number of faults remaining in the software at time t and hence has to be a monotonically non-increasing function of time. Hence the nature of the failure intensity, $\lambda(t)$, is governed by the nature of failure occurrence rate per fault i.e. $s(t)$.

If we incorporate the effect of testing efficiency i.e. possibility of imperfect fault removal with p as the probability of perfect debugging and error generation during the debugging of observed faults with a constant fault introduction rate α, the

general model under perfect debugging environment can be modified accordingly. The total number of faults present at any moment of testing time, say $a(t)$ is a function of time and can be expressed as a linear function of the expected number of faults detected by time t, i.e.

$$a(t) = a + \alpha m(t) \tag{6.4.4}$$

Hence the intensity function of the generalized model under imperfect debugging environment becomes

$$\lambda(t) = m'(t) = \frac{dm(t)}{dt} = (a(t) - m(t))p\frac{F'(t)}{1 - F(t)} \tag{6.4.5}$$

Substituting for $a(t)$ in (6.4.5) and solving under the initial condition that at $t = 0$, $m(0) = 0$, we obtain

$$m(t) = \frac{a}{1-\alpha}\left[1 - (1 - F(t))^{p(1-\alpha)}\right] \tag{6.4.6}$$

The mean value function in (6.4.6) represents the expected number of faults detected/corrected for the generalized SRGM incorporating the effect of testing efficiency under the assumption of immediate removal of faults on failure observation (GINHPP-1). Now we can obtain mean value functions of the various existing and several new SRGM from (6.4.6) using the different forms of the distribution function $F(t)$.

Now we will show how to obtain existing as well as new models from the GINHPP-1. Suppose we assume that an exponential distribution function describes the $F(t)$ i.e.

$$F(t) = 1 - \mathrm{e}^{-bt} \tag{6.4.6}$$

then

$$\frac{F'(t)}{1 - F(t)} = b$$

it implies

$$m(t) = \frac{a}{1-\alpha}\left[1 - \mathrm{e}^{-bp(1-\alpha)t}\right] \tag{6.4.7}$$

The mean value function (6.4.7) describes the imperfect debugging model given by Kapur et al. [14] defining imperfect debugging and error generation. For this model when $t \to \infty$, $m(t) \to \frac{a}{1-\alpha}$ which implies that if testing is carried out for an infinite time, more faults are removed as compared to the initial fault content because some faults are manifested in the software due to error generation during the debugging activity. If $p = 1$ and $\alpha = 0$, the case of no error generation and perfect repair, we obtain the pure perfect debugging exponential GO model due to Goel and Okumoto [11]. Similarly for the distinct distribution functions $F(t)$ different models can be obtained. The mean value functions $m(t)$ of several SRGM

corresponding to different forms of distribution functions $F(t)$ are summarized in Table 6.3.

Model M2 is the imperfect debugging model given by Kumar et al. [30] defining imperfect debugging and error generation. For this model $\frac{F'(t)}{1-F(t)} = \frac{b^2 t}{1+bt}$which is the hazard rate of the Yamada delayed S-shaped [17] perfect debugging model which is obtainable if $p = 1$ and $\alpha = 0$, i.e. perfect debugging. In model M3 if $k = 3$, the mean value function reduces to $m(t) = \frac{a}{1-\alpha}\left[1 - \left(\left(1 + bt + \frac{b^2t^2}{2}\right)e^{-bt}\right)^{p(1-\alpha)}\right]$. In this model if we substitute $p = 1$ and $\alpha = 0$, we have an SRGM expressed by three-stage Erlang growth curve [18]. Model M4 is a generalized imperfect debugging model accounting for the experience gained by the testing team as time and testing progresses. A major advantage of following this unification scheme comes from the fact that we can obtain the mean value functions of the SRGM with Weibull, Gamma and Normal correction times under imperfect debugging environment. The model expressions obtained in M5, M6 and M7 are rather not obtainable if we follow the usual procedure of formulating an imperfect debugging model as the differential equation which can describe the physical form of these models becomes very complex.

6.4.2 *Generalized SRGM Considering Time Delay Between Failure Observation and Correction Procedures Under Imperfect Debugging Environment*

To incorporate the concept of FDCP with a delay the unification scheme discussed above is further generalized with the two distribution functions $F(t)$ and $G(t)$. The distribution $F(t)$ defines the failure detection and $G(t)$ defines the correction

Table 6.3 SRGM obtained from unification scheme in Sect. 6.4.1

Model	Distribution function ($F(t)$)	Mean value function $m(t)$
M1	An exponential distribution $1 - e^{-bt}$	$\frac{a}{1-\alpha}\left[1 - e^{-bp(1-\alpha)t}\right]$
M2	Two-stage Erlang distribution $1 - (1 + bt)e^{-bt}$	$\frac{a}{1-\alpha}\left[1 - \left((1 + bt)e^{-bt}\right)^{p(1-\alpha)}\right]$
M3	k-stage Erlang distribution $1 - \left(\sum_{i=0}^{k-1} (bt)^i / i!\right)e^{-bt}$	$\frac{a}{1-\alpha}\left[1 - \left(\left(\sum_{i=0}^{k-1} (bt)^i / i!\right)e^{-bt}\right)^{p(1-\alpha)}\right]$
M4	$P(t) = 1 - \left(\frac{\left(\beta + \sum_{i=0}^{k-1} \frac{(bt)^i}{i!}\right)e^{-bt}}{(1+\beta e^{-bt})}\right)$	$\frac{a}{1-\alpha}\left[1 - \left(\frac{\left(\beta + \sum_{i=0}^{k-1} \frac{(bt)^i}{i!}\right)e^{-bt}}{(1+\beta e^{-bt})}\right)^{p(1-\alpha)}\right]$
M5	Weibull distribution $T \sim Wei(b, k)$	$\frac{a}{1-\alpha}\left[1 - e^{-bp(1-\alpha)t^k}\right]$
M6	Normal distribution $T \sim N(\mu, \sigma^2)$	$\frac{a}{1-\alpha}\left[1 - (1 - \varphi(t, \mu, \sigma))^{p(1-\alpha)}\right]$
M7	Gamma distribution $T \sim \gamma(\alpha_1, \beta_1)$	$\frac{a}{1-\alpha}\left[1 - (1 - \Gamma(t, \alpha_1, \beta_1))^{p(1-\alpha)}\right]$

processes and the delay between the two process is described using the Stieltjes convolution. Hence the mean value function of the generalized model expressed in (6.4.1) is modified as (on the lines of Musa et al. [24])

$$m(t) = a(F \otimes G)(t) \tag{6.4.8}$$

The intensity function $\lambda(t)$ is given by

$$\lambda(t) = \frac{dm(t)}{dt} = a[(F \otimes G)(t)]' = a(f * g)(t) \tag{6.4.9}$$

The above equation can be rewritten as

$$\frac{dm(t)}{dt} = [a - m(t)]\frac{(f * g)(t)}{[1 - (F \otimes G)(t)]} \tag{6.4.10}$$

or

$$\frac{dm(t)}{dt} = h(t)[a - m(t)]$$

where $h(t) = \frac{(f*g)(t)}{[1-(F\otimes G)(t)]}$ is the failure observation/fault correction rate.

Now incorporating the concepts of imperfect debugging and error generation in the manner similar to (6.4.5) we have

$$\frac{dm(t)}{dt} = \frac{(f * g)(t)}{[1 - (F \otimes G)(t)]}p[a + \alpha m(t) - m(t)] \tag{6.4.11}$$

Solving the above differential equation, we get the final exact solution

$$m(t) = \frac{a}{(1 - \alpha)}\left[1 - (1 - (F \otimes G)(t))^{p(1-\alpha)}\right] \tag{6.4.12}$$

Mean value function in (6.4.12) is the generalized SRGM considering time delay between failure observation and correction procedures under imperfect debugging environment. Using this generalized model we can obtain the mean value functions of the several existing and new SRGM distinguishing FDCP. The mean value functions $m(t)$ corresponding to different forms of distribution functions $F(t)$ and $G(t)$ are summarized in Table 6.4.

In the above models if we substitute $p = 1$ and $\alpha = 0$ we obtain the corresponding SRGM under perfect debugging environment. With this statement it follows that we can call this unification scheme due to Kapur et al. [13] as the unification scheme for all the other unification schemes discussed up to now due to the reason that we can obtain almost all of the existing SRGM both defined under perfect and imperfect debugging environment from it. It makes it very important to build a thorough understanding of this unification scheme for the software engineers and software reliability practitioners.

Table 6.4 SRGM obtained from unification scheme in Sect. 6.4.2

Model	$F(t)$	$G(t)$	$m(t)$
M8	$t \sim \exp(b)$	$1(t)$	$\frac{a}{1-\alpha}\left[1-\mathrm{e}^{-bp(1-\alpha)t}\right]$
M9	$t \sim \exp(b)$	$t \sim \exp(b)$	$\frac{a}{1-\alpha}\left[1-\left((1+bt)\mathrm{e}^{-bt}\right)^{p(1-\alpha)}\right]$
M10	$t \sim \exp(b)$	$t \sim \exp(b_2)$	$\frac{a}{1-\alpha}\left[1-\left\{\frac{1}{b_1-b_2}\begin{pmatrix} b_1\mathrm{e}^{-b_2 t} \\ -b_2\mathrm{e}^{-b_1 t}\end{pmatrix}\right\}^{p(1-\alpha)}\right]$
M11	$t \sim \text{Erlang}-2(b)$	$t \sim \exp(b)$	$\frac{a}{1-\alpha}\left[1-\left(\left(1+bt+\frac{b^2t^2}{2}\right)\mathrm{e}^{-bt}\right)^{p(1-\alpha)}\right]$
M12	$t \sim \exp(b)$	$t \sim N(\mu,\sigma^2)$	$\frac{a}{1-\alpha}\left[1-\left(\begin{matrix}1-\varphi(t,\mu,\sigma)+ \\ \left(\begin{matrix}\varphi\left(t,\mu+b\sigma^2,\sigma\right) \\ \exp\left(-bt+\mu b+\frac{(b\sigma)^2}{2}\right)\end{matrix}\right)\end{matrix}\right)^{p(1-\alpha)}\right]$
M13	$t \sim \exp(b)$	$t \sim \gamma(\alpha_1,\beta_1)$	$\frac{a}{1-\alpha}\left[1-\left(\begin{matrix}1-\Gamma(t,\alpha_1,\beta_1)+ \\ \left(\begin{matrix}\frac{\mathrm{e}^{-\mathrm{bt}}}{(1-b\beta_1)^{\alpha_1}} \\ \Gamma\left(t,\alpha_1,\frac{\beta_1}{1-b\beta_1}\right)\end{matrix}\right)\end{matrix}\right)^{p(1-\alpha)}\right]$
M14	$t \sim U(0,1)$	$t \sim Weib(\alpha,m)$	$\frac{a}{1-\alpha}\left[1-\left(1-\left(\begin{matrix}t- \\ \frac{1}{m\alpha_1}\left\{\begin{matrix}m\Gamma_1\left(1+\frac{1}{m}\right) \\ -\Gamma_2\left(\begin{matrix}\frac{1}{m}, \\ (\alpha t)^m\end{matrix}\right)\end{matrix}\right\}\end{matrix}\right)\right)^{p(1-\alpha)}\right]$

where Γ_1 is a Gamma function; Γ_2 is an upper incomplete Gamma function

6.5 An Equivalence Between the Three Unified Approaches

In this chapter we have discussed three unification methodologies

1. Unification of SRGM for FDCP [23].
2. Infinite server queuing methodology [12].
3. A unified approach in the presence of imperfect debugging and error generation [13].

Recently Kapur et al. [14] have shown that although these unifying schemes, derived under different sets of assumptions, are mathematically equivalent.

The unification methodology of infinite server Queues for the hard faults, fault-detection correction process with a delay function and one based on detection correction using the hazard function concept under perfect debugging environment is proved equivalent by them.

6.5.1 Equivalence of Unification Schemes Based on Infinite Server Queues for the Hard Faults and Fault Detection Correction Process with a Delay Function

Consider the unification methodology of Xie et al. [23] based on the concept of time lag between fault-detection and correction, where (refer to Eq. (6.2.2)),

$$m_c(t) = \int_0^t \lambda_c(s)\,\mathrm{d}s = \int_0^t E[\lambda_d(s - \Delta(s))]\,\mathrm{d}s \tag{6.5.1}$$

If $f(x)$ is the *pdf* of the random correction time then we have

$$\lambda_c(t) = E[\lambda_d(t - \Delta(t))] = \int_0^s \lambda_d(s - x) f(x)\,\mathrm{d}x \tag{6.5.2}$$

From (6.5.1) and (6.5.2) we have

$$\begin{aligned} m_c(t) &= \int_0^t \int_0^s \lambda_d(s - x) f(x)\,\mathrm{d}x\,\mathrm{d}s \\ &= \int_0^t \int_x^t \lambda_d(s - x)\,\mathrm{d}s f(x)\,\mathrm{d}x \end{aligned} \tag{6.5.3}$$

$$m_c(t) = \int_0^t m_d(t-x) f(x)\,\mathrm{d}x \tag{6.5.4}$$

$$m_c(t) = \int_0^t F(t-x)\,\mathrm{d}m_d(x) \tag{6.5.5}$$

which is same as (6.3.16), the unified SRGM for the hard faults based on the concept of infinite queues. It may also be noted here that for obtaining the SRGM for detection and correction process only the unified SRGM for hard faults needs to be considered for the unification scheme in Sect. 6.3.

6.5.2 *Equivalence of Unification Schemes Based on Infinite Server Queues for the Hard Faults and One Based on Hazard Rate Concept*

The next step establishes the equivalence of infinite server queuing model to unification scheme based on hazard rate [13].

Consider Eq. (6.5.5)

$$\begin{aligned} m_c(t) &= \int_0^t F(t-x)\,\mathrm{d}m_d(x) = F(t) \otimes m_d(t) \\ &= \int_0^t F_c(t-x)\,\mathrm{d}m_d(x) \\ &= F_c(t) \otimes m_d(t) \end{aligned} \tag{6.5.6}$$

Now using (6.4.1) we have $m_d(t) = aF_d(t)$

$$\Rightarrow m_c(t) = a(F \otimes F_d)(t) = a(F_d \otimes F)(t) \tag{6.5.7}$$

which is the same as (6.4.8), the unification scheme based on the hazard rate (Sect. 6.4) under perfect debugging environment. From (6.5.7) it follows that the three unification schemes discussed in this chapter are mathematically equivalent.

6.6 Data Analysis and Parameter Estimation

As we have learned that the development of unification schemes for SRGM development and application makes it easy for the practitioners to apply SRGM in

practice. Several models with different characteristics get a same structural interpretation and a single approach for the development of various SRGM enables the non-mathematical practitioners to conveniently select diverse types of SRGM and select the best for their particular application. Several existing and new SRGM are developed through the three unification schemes discussed in the chapter. Data analysis of many of them is already discussed in the previous chapters. Here we have discussed the application of some new SRGM developed through the unification methodology.

6.6.1 Application of SRGM for Fault Detection and Correction Process

SRGM for FDCP can be obtained from the unification scheme for FDCP (Sect. 6.2) and testing efficiency based software reliability modeling (Sect. 6.4). Here we have chosen some models discussed in both of the sections.

Failure Data Set

The software-testing data sets reported in the literature are obtained generally from the failure process. Xie et al. [23] reported a joint software-testing data for both failure observation and correction. The data set is from the testing process on a middle-size software project grouped in number of faults per week. The testing data are for 17 weeks during which 144 faults were observed and 143 of them are corrected. The fault correction process seems to be slow in the beginning for three weeks which picked up afterwards.

Following models are chosen for data analysis and parameter estimation. The failure observation process of all these models except model M7 is described by the GO model [11], i.e.

$$m_d(t) = a\left(1 - \mathrm{e}^{-bt}\right) \quad \text{or} \quad a\left(1 - \mathrm{e}^{-b_1 t}\right)$$

For the model M7 detection process is described by the two-stage Erlang distribution.

$$m_d(t) = a\left(1 - (1 + bt)\mathrm{e}^{-bt}\right)$$

Model 1 (M1) Constant correction time SRGM [23]

$$m_c(t) = m_d(t - \Delta) = a\left(1 - \mathrm{e}^{-b(t-\Delta)}\right); \quad t \geq \Delta$$

Model 2 (M2) Time dependent correction time SRGM [23]

$$m_c(t) = a\left(1 - (1 + ct)\mathrm{e}^{-bt}\right)$$

Model 3 (M3) Exponentially distributed correction time SRGM [23]

$$m_c(t) = a\left(1 - \frac{\mu}{\mu - b}\mathrm{e}^{-bt} + \frac{b}{\mu - b}\mathrm{e}^{-\mu t}\right)\mu \neq b$$

Model 4 (M4) Normal correction time delay SRGM [23]

$$m_c(t) = -a\mathrm{e}^{\left(-bt+\mu b+(b\sigma)^2/2\right)}\begin{pmatrix}\Phi\left(t, b\sigma^2+\mu, \sigma\right)-\\ +\Phi\left(0, b\sigma^2+\mu, \sigma\right)\end{pmatrix} + a(\Phi(t, \mu, \sigma) - \Phi(0, \mu, \sigma))$$

Model 5 (M5) Gamma correction time delay SRGM [23]

$$m_c(t) = a\Gamma(t, \alpha, \beta) - \frac{a\mathrm{e}^{-\mathrm{bt}}}{(1 - b\beta)^{\alpha}}\Gamma\left(t, \alpha, \frac{\beta}{(1 - b\beta)}\right)$$

Model 6 (M6) Exponentially distributed correction time, testing efficiency based SRGM [13]

$$m_c(t) = \frac{a}{1-\alpha}\left[1 - \left\{\frac{1}{b_1 - b_2}\begin{pmatrix} b_1\mathrm{e}^{-b_2 t}\\ -b_2\mathrm{e}^{-b_1 t}\end{pmatrix}\right\}^{p(1-\alpha)}\right]$$

Model 7 (M7) Two-stage Erlang type detection process with exponentially distributed correction time, testing efficiency based SRGM [13]

$$m_c(t) = \frac{a}{1-\alpha}\left[1 - \left(\left(1 + bt + \frac{b^2t^2}{2}\right)\mathrm{e}^{-bt}\right)^{p(1-\alpha)}\right]$$

Model 8 (M8) Normal delay correction time, testing efficiency based SRGM [13]

$$m_c(t) = \frac{a}{1-\alpha}\left[1 - \left(1 - \varphi(t, \mu, \sigma) + \begin{pmatrix}\varphi\left(t, \mu + b\sigma^2, \sigma\right)\\ \exp\left(-bt + \mu b + \frac{(b\sigma)^2}{2}\right)\end{pmatrix}\right)^{p(1-\alpha)}\right]$$

Model 9 (M9) Gamma delay correction time, testing efficiency based SRGM [13]

$$m_c(t) = \frac{a}{1-\alpha}\left[1 - \left(1 - \Gamma(t, \alpha_1, \beta_1) + \begin{pmatrix}\frac{\mathrm{e}^{-bt}}{(1-b\beta_1)^{\alpha_1}}\\ \Gamma\left(t, \alpha_1, \frac{\beta_1}{1-b\beta_1}\right)\end{pmatrix}\right)^{p(1-\alpha)}\right]$$

The results of parameter estimations are listed in Table 6.5 and the goodness of fit curves for the fault observation and correction process for models M1–M5 are shown in Figs. 6.2 and 6.3, respectively, and the goodness of fit curves for the imperfect debugging models M6–M9 for observation and correction process are shown, respectively, in Figs. 6.4 and 6.5.

Table 6.5 Estimation results of models M1–M9

Models	Estimated parameters								Comparison criteria		
	a	α	β, β_1	b, b_1	b_2, c	μ, α_1	σ	p	MSEd	MSEc	R^2
M1	178	–	–	0.0999	1.0000	–	–	–	87.48	86.99	0.967
M2	168	–	–	0.1193	0.0279	–	–	–	65.83	184.20	0.942
M3	156	–	–	0.1404	–	0.5811	–	–	57.39	72.55	0.979
M4	149	–	–	0.1790	–	2.1987	1.0283	–	117.61	26.75	0.991
M5	150	16.69	0.1169	0.1537	–	–	–	–	58.98	40.43	0.988
M6	145	0.0202	–	0.2522	0.2676	–	–	0.9901	413.60	70.59	0.979
M7	136	0.0214	–	0.4517	–	–	–	0.9840	83.46	36.38	0.988
M8	164	0.0769	–	0.2002	–	1.9012	1.7430	0.9195	657.72	41.24	0.988
M9	143	0.0155	0.5733	0.1834	–	1.9179	–	0.9823	85.30	22.25	0.992

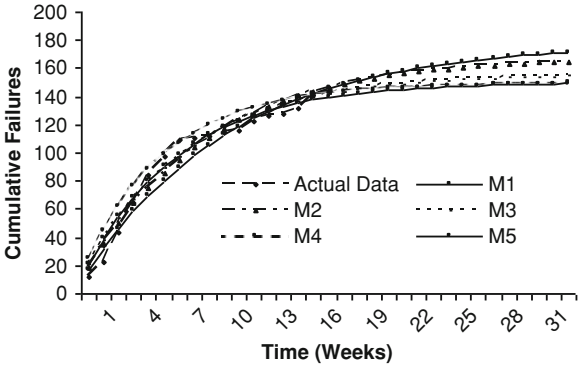


Fig. 6.2 Goodness of fit curve of detection process for models M1–M5

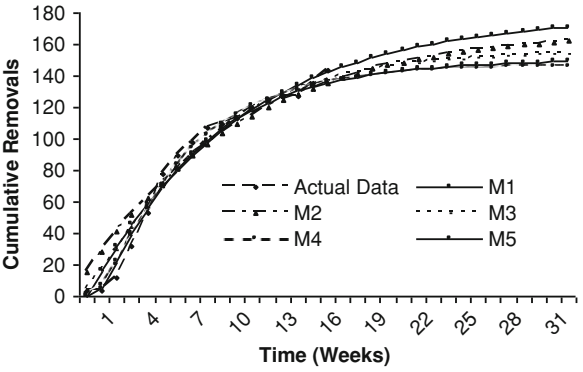


Fig. 6.3 Goodness of fit curve of correction process for models M1–M5

The software-testing data corresponding to the correction process are used here to fit the SRGM for the correction processes. Using the estimates of the correction process (parameter a, b) we have estimated the detection process.

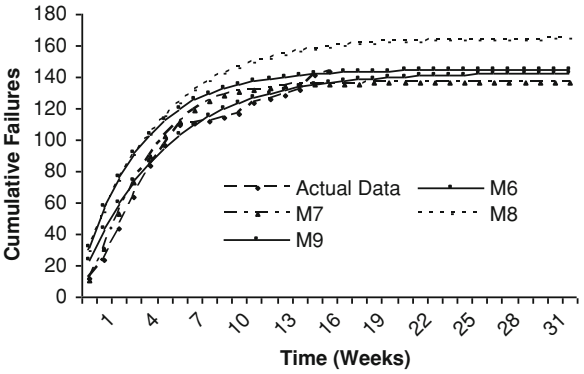


Fig. 6.4 Goodness of fit curve of detection process for models M6–M9

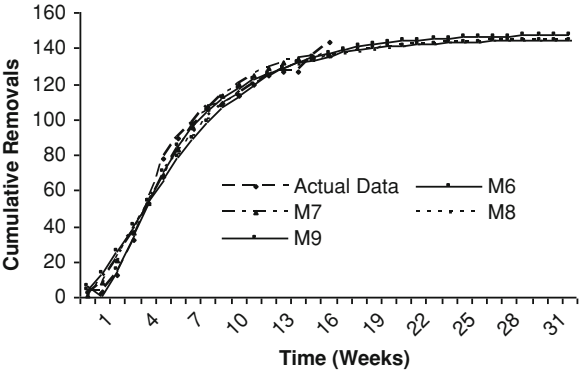


Fig. 6.5 Goodness of fit curve of correction process for models M6–M9

Here the R^2 figures are corresponding to the data analysis of the fault correction process. We have calculated the mean square errors for both the detection (MSEd) and correction (MSEc) process. Results in Table 6.5 depict that the correction process is best described by the testing efficiency based Gamma correction delay time model (M9), but the Mean Square errors (MSE) corresponding to the detection process for this model is higher compared to the other models. On the other hand if we see the result of model M5 which is also based on Gamma distributed correction delay time model, but assumes perfect debugging, we can say that this model can be chosen for the analysis of the testing process of this software project. Although the MSE for the correction process is higher for this model compared to M9 it has better value of MSE for the detection process and both of the MSE are comparable. However in such a case it remains the subjective choice of the practitioners to decide which models are to use depending on their own testing and environmental profile.

6.6.2 Application of SRGM Based on the Concept of Infinite Server Queues

Failure Data Set

The interval domain data is taken from Misra [31] in which the number of faults detected per week (38 weeks) is specified and a total of 231 faults were detected. Three types of faults—critical (1.73%), major (34.2%) and minor (64.07%), are present in the software.

The following models have been selected for illustrating the data analysis and parameter estimation. In the following models p_1, p_2 and p_3 are the proportion of complex, hard and simple faults, respectively. Here we have chosen the fault complexity models although the technique of infinite server queues can also be used for the development of models for the one type faults. We have discussed in Sect. 6.5 that each of the three unification scheme discussed in this chapter are equivalent. The data analysis for the new models of single types developed using this scheme is already discussed in the previous section.

Model 10 (M10) Fault complexity SRGM where fault removal process for each fault is described by GO model with different Fault Removal Rate (FRR) [12]

$$m(t) = ap_1\left(1 - e^{-b_1 t}\right) + ap_2\left(1 - e^{-b_2 t}\right) + ap_3\left(1 - e^{-b_3 t}\right), \quad p_1 + p_2 + p_3 = 1$$

Model 11 (M11) Generalized Erlang SRGM with same FRR for each type of fault [12]

$$m(t) = ap_1\left(1 - \left(1 + bt + \frac{b^2t^2}{2}\right)e^{-bt}\right) + ap_2\left(1 - (1 + bt)e^{-bt}\right) + ap_3\left(1 - e^{-bt}\right), \quad p_1 + p_2 + p_3 = 1$$

Model 12 (M12) Generalized Erlang SRGM with different FRR for each type of fault [12]

$$m(t) = ap_1\left(1 - \left(1 + b_1 t + \frac{b_1^2 t^2}{2}\right)e^{-b_1 t}\right) + ap_2\left(1 - (1 + b_2 t)e^{-b_2 t}\right) + ap_3\left(1 - e^{-b_3 t}\right), \quad p_1 + p_2 + p_3 = 1$$

Model 13 (M13) Three types of faults and FRP are described by GO model for simple, delayed S-shaped for hard and Gamma time delay model for complex faults [12]

$$m(t) = ap_1\left(\Gamma(t, \alpha, \beta) - \left(\frac{e^{-b_1 t}}{(1 - b_1\beta)^{\alpha}}\Gamma\left(t, \alpha, \frac{\beta}{(1 - b_1\beta)}\right)\right)\right) + ap_2\left(1 - (1 + b_2 t)e^{-b_2 t}\right) + ap_3\left(1 - e^{-b_3 t}\right), \quad p_1 + p_2 + p_3 = 1$$

Table 6.6 Estimation results of models M10–M14

Models	Estimated parameters						Comparison criteria	
	a	μ, α	σ, β	b, b_1	b_2	b_3	MSE	R^2
M10	675	–	–	0.0099	0.0098	0.8580	14.30	0.992
M11	413	–	–	0.0286	–	–	20.91	0.994
M12	655	–	–	0.0184	0.0065	0.0344	19.83	0.995
M13	511	1.7633	0.0086	0.0237	0.0168	0.0270	19.89	0.995
M14	569	1.1660	0.0059	0.0210	0.0141	0.0001	20.16	0.999

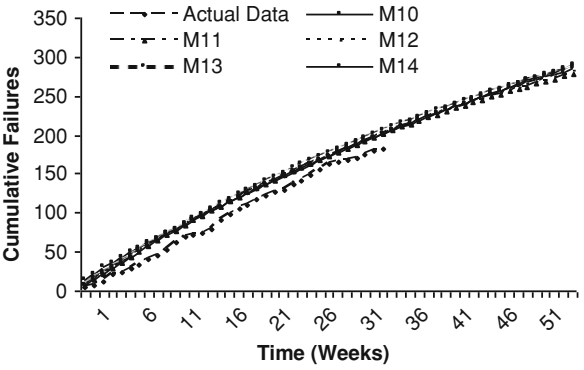


Fig. 6.6 Goodness of fit curve for models M10–M14

Model 14 (M14) Three types of faults and FRP are described by GO model for simple, delayed S-shaped for hard and normal time delay model for complex faults [12]

$$m(t) = ap_1\left(\left(-\Phi\big((0,t), b_1\sigma^2 + \mu, \sigma\big)\mathrm{e}^{\left(-b_1 t + \mu b_1 + \frac{(b_1\sigma)^2}{2}\right)}\right) + \Phi((0,t), \mu, \sigma)\right)$$
$$+ ap_2\big(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\big) + ap_3\big(1 - \mathrm{e}^{-b_3 t}\big), \quad p_1 + p_2 + p_3 = 1$$

The results of regression analysis of models M10–M14 are listed in Table 6.6 and the goodness of fit curve against the actual data is shown in Fig. 6.6. From the table we can conclude that model M10 fits best on this data set. This model describes the removal process for each type of fault by the exponential models with different fault removal rates. It means that the software is tested under a uniform operational profile and the complexity of each type of fault can be described similarly with different values of parameters. Another interpretation of the results is that the removal rate of simple faults is quite high as compared to the hard and complex faults. On the other hand removal rate for the other two types of faults is similar, indicating the presence of only two types of faults in the system.

6.6.3 Application of SRGM Based on Unification Schemes for Testing Efficiency Models

Two types of testing efficiency SRGM can be developed using the unification schemes for the testing efficiency based SRGM. First the SRGM where the detection process is assumed to describe the removal process also or in other words the SRGM which assumes no time delay in fault removal after detection. The second type of SRGM where the detection and removal process is described by the different model equations and it is assumed that the removal process defers the detection process. Application of the second type of SRGM is already discussed in Sect. 6.6.3. Now we show the application of the first type SRGM in this section.

Failure Data Set

This data set was collected in the bug tracking system on the website of Xfce [32]. Xfce is a lightweight desktop environment for UNIX-like OS. The observation period for the data is 21 weeks and during the 21 weeks of testing 167 faults was observed.

The following models have been selected for illustrating the data analysis and parameter estimation.

Model 15 (M15) Exponential imperfect debugging model [13]

$$\frac{a}{1-\alpha}\left[1-\mathrm{e}^{-bp(1-\alpha)t}\right]$$

Model 16 (M16) Two-stage Erlang distribution based imperfect debugging model [13]

$$\frac{a}{1-\alpha}\left[1-\mathrm{e}^{-bp(1-\alpha)t}\right]$$

Model 17 (M17) Weibull distribution based imperfect debugging model [13]

$$\frac{a}{1-\alpha}\left[1-\mathrm{e}^{-bp(1-\alpha)t^k}\right]$$

Model 18 (M18) Normal distribution based imperfect debugging model [13]

$$\frac{a}{1-\alpha}\left[1-(1-\varphi(t,\mu,\sigma))^{p(1-\alpha)}\right]$$

Model 19 (M19) Normal distribution based imperfect debugging model [13]

$$\frac{a}{1-\alpha}\left[1-(1-\Gamma(t,\alpha_1,\beta_1))^{p(1-\alpha)}\right]$$

The results of regression analysis of models M15–M19 are listed in Table 6.7 and the goodness of fit curve against the actual data is shown in Fig. 6.7. From the

Table 6.7 Estimation results for models M10–M14

Models	Estimated parameters							Comparison criteria	
	a	α	b, b_1,	b_2, c, k	μ, α_1	σ, β_1	p	MSE	R^2
M15	470	0.0399	0.0219	–	–	–	0.9241	23.05	0.991
M16	176	0.0271	0.1746	–	–	–	0.9545	78.98	0.971
M17	405	0.0233	0.0237	1.0172	–	–	0.9734	25.10	0.988
M18	498	0.0202	–	–	1.0120	21.86	0.9780	29.79	0.986
M19	514	0.0323	–	–	0.9782	0.0180	0.9665	24.20	0.986

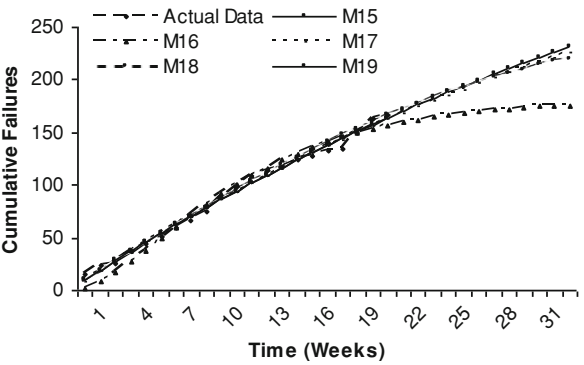


Fig. 6.7 Goodness of fit curve for models M15–M19

table we can conclude that model M10 fits best on this data set. This model describes the removal process for each type of fault by the exponential models with different fault removal rates. It means that the software is tested under a uniform operational profile and the complexity of each type of fault can be described similarly with different values of parameters. Another interpretation of the results is that the removal rate of simple faults is quite high as compared to the hard and complex faults. On the other hand removal rate for the other two types of faults is similar, indicating the presence of only two types of faults in the system.

Exercises

1. Why unification in software reliability growth modeling have been developed?
2. Assume FDP of a software can be described by the mean value function of an exponential SRGM, i.e. $m_f(t) = a\left(1 - \mathrm{e}^{-bt}\right)$ and the fault isolation and removal times are assumed to be independent with a common distribution $G(t)$, with pdf $g(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\Pi}}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Obtain the mean value function of the SRGM for the isolation and removal process using the infinite server queue based unification technique.

3. If the distribution of failures and removal of faults is exponential with parameters b_1 and b_2, respectively, then show using the unification technique discussed in Sect. 6.4.2 the mean value function of SRGM is given as

$$\frac{a}{1-\alpha}\left[1-\left\{\frac{1}{b_1-b_2}\begin{pmatrix} b_1 e^{-b_2 t} \\ -b_2 e^{-b_1 t}\end{pmatrix}\right\}^{p(1-\alpha)}\right]$$

References

1. Shanthikumar JG (1981) A general software reliability model for performance prediction. Microelectron Reliab 21(5):671–682
2. Shanthikumar JG (1983) Software reliability models: a review. Microelectron Reliab 23(5):903–943
3. Langberg N, Singpurwalla ND (1985) Unification of some software reliability models. SIAM J Comput 6:781–790
4. Miller DR (1986) Exponential order statistic models of software reliability growth. IEEE Trans Softw Eng SE-12:12–24
5. Thompson WA Jr (1988) Point process models with applications to safety and reliability. Chapman and Hall, New York
6. Gokhale SS, Philip T, Marinos PN, Trivedi KS (1996) Unification of finite failure non-homogeneous Poisson process models through test coverage. In: Proceedings 7th international symposium on software reliability engineering, White Plains, pp 299–307
7. Chen Y, Singpurwalla ND (1997) Unification of software reliability models by self-exciting point processes. Adv Appl Probab 29:337–352
8. Gokhale SS, Trivedi KS (1999) A time/structure based software reliability model. Ann Softw Eng 8(1–4):85–121
9. Huang CY, Lyu MR, Kuo SY (2003) A unified scheme of some non-homogeneous Poisson process models for software reliability estimation. IEEE Trans Softw Eng 29:261–269
10. Jelinski Z, Moranda P (1972) Software reliability research. In: Freiberger W (ed) Statistical computer performance evaluation. Academic Press, New York, pp 465–484
11. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R-28(3):206–211
12. Kapur PK, Anand S, Inoue S, Yamada S (2010) A unified approach for developing software reliability growth model using infinite server queuing model. Int J Reliab Qual Safety Eng, 17(5):401–424, doi No: 10.1142/S0218539310003871
13. Kapur PK, Pham H, Anand S, Yadav K (2011) A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans Softw Reliab, in press, doi:10.1109/TR.2010.2103590
14. Kapur PK, Aggarwal AG, Anand S (2009) A new insight into software reliability growth modeling. Int J Performability Eng 5(3):267–274
15. Schneidewind NF (1975) Analysis of error processes in computer software. Sigplan Not 10:337–346
16. Xie M, Zhao M (1992) The Schneidewind software reliability model revisited. In: Proceedings 3rd international symposium on software reliability engineering, pp 184–192
17. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R-32(5):475–484

18. Kapur PK, Younes S (1995) Software reliability growth model with error dependency. Microelectron Reliab 35(2):273–278
19. Huang CY, Lin CT (2006) Software reliability analysis by considering fault dependency and debugging time lag. IEEE Trans Reliab 35(3):436–449
20. Lo HJ, Huang CY (2006) An integration of fault-detection and correction processes in software reliability analysis. J Syst Softw 79:1312–1323
21. Singh VB, Yadav K, Kapur R, Yadavalli VSS (2007) Considering fault dependency concept with debugging time lag in software reliability growth modeling using a power function of testing time. Int J Autom Comput 4(4):359–368
22. Wu YP, Hu QP, Xie M, Ng SH (2007) Modeling and analysis of software fault-detection and correction process by considering time dependency. IEEE Trans Reliab 56(4):629–642
23. Xie M, Hu QP, Wu YP, Ng SH (2007) A study of the modeling and analysis of software fault-detection and fault-correction processes. Qual Reliab Eng Int 23:459–470
24. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York ISBN 0–07-044093-X
25. Luong B, Liu DB (2001) Resource allocation model in software development. In: Proceedings 47th IEEE annual reliability and maintainability symposium, Philadelphia, USA, January 2001, pp 213–218
26. Antoniol G, Cimitile A, Lucca GA, Penta MD (2004) Assessing staffing needs for a software maintenance project through queuing simulation. IEEE Trans Softw Eng 30(1):43–58
27. Inoue S, Yamada S (2002) A software reliability growth modeling based on infinite server queuing theory. In: Proceedings 9th ISSAT international conference on reliability and quality in design, Honolulu, HI, pp 305–309
28. Dohi T, Osaki S, Trivedi KS (2004) An infinite server queuing approach for describing software reliability growth—unified modeling and estimation framework. In: Proceedings 11th Asia-Pacific software engineering conference (APSEC'04), pp 110–119
29. Ross, S.M. (1970) Applied probability models with optimization applications. Holden-Day, San Francisco
30. Kumar Deepak, Kapur R, Sehgal VK, Jha PC (2007) On the development of software reliability growth models with two types of imperfect debugging. Int J Communications in Dependability and Quality Management 10(3):105–122
31. Misra PN (1983) Software reliability analysis. IBM Syst J 22:262–270
32. Tamura Y, Yamada S (2005) Comparison of software reliability assessment methods for open source software. In: Proceedings 11th international conference on parallel and distributed systems (ICPADS 2005), Los Almitos, CA, USA, pp 488–492

Chapter 7
Artificial Neural Networks Based SRGM

7.1 Artificial Neural Networks: An Introduction

An Artificial Neural Network (ANN) is a computational paradigm that is inspired by the behavior of biological nervous system. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems capable of revealing complex global behavior, determined by the connections between the processing elements and element parameters. ANN, like people, learns by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. In more practical terms neural networks are non-linear statistical data modeling or decision making tools. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for ANN as well.

Neural network simulation appears to be a recent development. However, this field was established before the advent of computers and has survived at least one major setback and several eras. Many important advances have been boosted by the use of inexpensive computer emulations. A brief history [1] of the development of the neural networks can be described diving into several periods.

- *First Attempts*: There were some initial simulations using formal logic. McCulloch and Pitts [2] developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "*a* or *b*" and "*a* and *b*." Another attempt was using computer simulations by two groups [3, 4]. The first group (IBM researchers) maintained close contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a multidisciplinary trend which continues to the present day.

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_7,
© Springer-Verlag London Limited 2011

- *Promising and Emerging Technology*: Not only the Nero-science was influential in the development of neural networks, but psychologists and engineers also contributed to the progress of neural network simulations. Rosenblatt [5] stirred considerable interest and designed and developed the Perceptron. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. Another system was the adaptive linear element (ADALINE) which was developed by Widrow and Hoff [6]. The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron, it employed the Least-Mean-Squares (LMS) learning rule.
- *Period of Frustration and Disrepute*: Minsky and Papert [7] wrote a book in which they generalized the limitations of single layer Perceptrons to multilayered systems. In the book they said: "…our intuitive judgment that the extension (to multilayer systems) is sterile." The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.
- *Innovation*: Although public interest and available funding were minimal, several researchers continued working to develop neuromorphically based computational methods for problems such as pattern recognition. During this period several paradigms were generated. Carpenter and Grossberg [8] influence founded a school of thought which explores resonating algorithms. They developed the Adaptive Resonance Theory (ART) networks based on biologically plausible models. Klopf [9] developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis. Werbos [10] developed and used the back-propagation learning method, however several years passed before this approach was popularized. Back-propagation networks are probably the most well known and widely applied neural networks today. In essence, the back-propagation network is a Perceptron with multiple layers, a different threshold function in the artificial neuron, and a more robust and capable learning rule. Anderson and Kohonen developed associative techniques independent of each other. Amari [11–13] was involved with theoretical developments: he published a paper which established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification.
- *Re-Emergence*: Progress during the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology. Academic programs appeared and courses were introduced at most major Universities (in USA and Europe). Attention is now focused on funding levels throughout Europe, Japan and the USA and as this funding

becomes available, several new commercial applications in industry and financial institutions are emerging.
- *Today*: Significant progress has been made in the field of neural networks, enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neurally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

Notations

$x(t)(x_i(t))$	The input to the hidden layer (ith neuron of hidden layer)
$h(t)(h_i(t))$	Output from the hidden layer (ith neuron of hidden layer)
$\alpha(x)(\alpha_i(x))$	Activation function in the hidden layer (ith neuron of hidden layer)
$y(t)$	The input to the output layer
$g(t)$	Output from the network (output layer)
$\beta(x)$	The activation function in the output layer
$w_1(w_{1i})$	Weights assigned to the input to the hidden layer
$w_2(w_{2i})$	Weights assigned to the input to the output layer
$b_1(b_{1i})$	Bias in the hidden layer
$b_0(b_{0i})$	Bias in the output layer

7.1.1 Specific Features of Artificial Neural Network

Neural networks find application because of their remarkable ability to derive meaning from complicated or imprecise data. These can be used to extract patterns and detect trends that are too complex to be noticed by either human beings or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real time operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

4. Fault tolerance via redundant information coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

7.2 Artificial Neural Network: A Description

In general, neural networks consist of three components [14]

1. Neurons
2. Network architecture
3. Learning algorithm

7.2.1 Neurons

An artificial neuron is a device with many inputs and one output. Neurons receive input signals, process the signals and finally produce an output signal. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not. Figure 7.1 shows a neuron, where f is the activation function that processes the input signals and produces an output of the neuron, x_i are the inputs of the neuron which may be the outputs from the previous layers, and w_i are the weights connected to the neurons of the previous layer.

7.2.2 Network Architecture

Artificial neural network is an interconnected group of artificial neurons that uses a computational model for information processing based on a connectionist

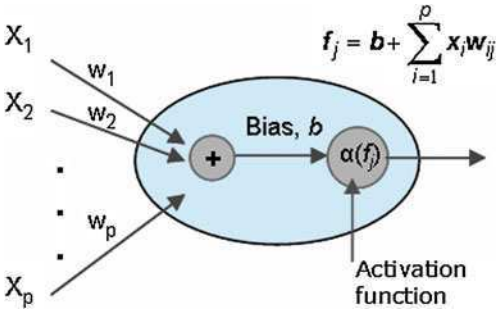


Fig. 7.1 A neuron

approach. An adaptive ANN changes its structure based on information that flows through the network. In general there are two most common types of neural network architectures—feed-forward networks and feedback networks.

A typical feed-forward neural network comprises a layer of neurons called *input layer* that receive inputs (suitably encoded) from the outside world, a layer called *output layer* that sends outputs to the external world, and one or more layers called *hidden layers* that have no direct communication with the external world. This hidden layer of neurons receives inputs from the previous layer and converts them to an activation value that can be passed on as input to the neurons in the next layer. The input layer neurons do not perform any computation; they merely copy the input values and associate them with weights, feeding the neurons in the first hidden layer. The input corresponds to the attributes measured for each training sample. The number of hidden layers is arbitrary. The weighted outputs of last hidden layer are input to units making up the output layer, which emits the network's prediction for given samples. An example of such a feed-forward network is shown in Fig. 7.2. In this figure there are p input units, q hidden units and r output units. Feed-forward networks can propagate activations only in the forward direction. There is no feedback (loops), i.e., the output of any layer does not affect that same layer. Feed-forward ANN tends to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down. On the other hand feedback networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their "state" is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the

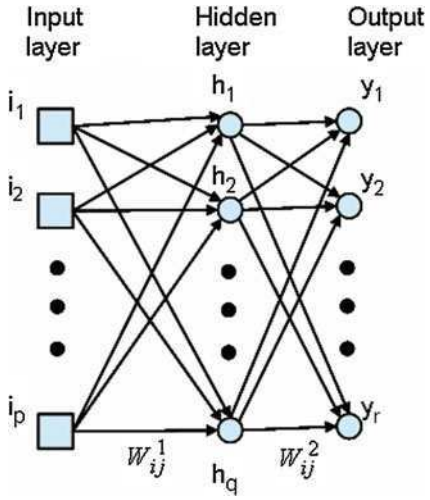


Fig. 7.2 A multilayer feed-forward neural network

latter term is often used to denote feedback connections in single-layer organizations.

7.2.3 Learning Algorithm

The learning algorithm describes a process to adjust the weights [1]. During the learning processes, the weights of network are adjusted to reduce the errors of the network outputs as compared to the standard answers. We can teach a three-layer network to perform a particular task by using the following procedure.

1. We present the network with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
2. We determine how closely the actual output of the network matches the desired output.
3. We change the weight of each connection so that the network produces a better approximation of the desired output.

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms

Associative mapping in which the network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms:

- *Auto-association*. An input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern completion, i.e., to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns.
- *Hetero-association*. Is related to two recall mechanisms:
  - *nearest-neighbor* recall, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
  - *interpolative* recall, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping, is classification, i.e., when there is a fixed set of categories into which the input patterns are to be classified.

Regularity detection in which, units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular "meaning." This type of learning mechanism is essential for feature discovery and knowledge representation.

Every neural network possesses knowledge which, is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. Information is stored in the weight matrix of a neural network. Learning is the determination of the weights. Following the way learning is performed, we can distinguish two major categories of neural networks

- *Fixed networks* in which the weights cannot be changed. In such networks, the weights are fixed a priori according to the problem to solve.
- *Adaptive networks* which are able to change their weights.

All learning methods used for adaptive neural networks can be classified into two major categories:

Supervised learning which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. An important issue concerning supervised learning is the problem of error convergence, i.e., the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square (LMS) convergence.

Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are: Hebbian learning and competitive learning.

We say that a neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back-propagation algorithm is the most widely used method for determining the EW and is also adopted for training the ANN discussed in this chapter. *Back propagation* is a supervised learning technique used for training ANNs. It was first described by Werbos [10], but the algorithm has been rediscovered a number of times. It is most useful for feed-forward networks. In back-propagation algorithm, the weights of the network are iteratively trained with the errors propagated back from the output layer. Back propagation learns by iteratively processing a set of training samples, comparing the network's prediction for each sample with the actual known value. For each training sample, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual value. It uses the

gradient of the sum-squared error (with respect to weights) to adapt the network weights so that the error measure is smaller in future epochs. The method requires that the transfer function used by the artificial neurons (or "nodes") be differentiable. Training terminates when the sum-squared error is below a specified tolerance limit.

The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection. Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the EA must be converted into the EI, the rate at which the error changes as the total input received by a unit is changed. Back propagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited. For software reliability modeling cumulative execution time is used as input and the corresponding cumulative faults as the desired output to form a training pair. Here the units of the network are non linear as most of the software reliability models describe nonlinear mathematical forms. The neural network can be described in a mathematical form. The objective of neural networks is to approximate a non-linear function that can receive the input vector $(x_1, x_2, \ldots, x_p)$in R^p and output the vector $(y_1, \ldots, y_r)$ in R^r.

Thus, the network can be denoted as:

$$y = \beta(i) \tag{7.2.1}$$

where $i = \left(i_1, i_2, i_3, \ldots, i_p\right)$ and $y = (y_1, y_2, y_3, \ldots, y_r)$. The value of any y_k is given by

$$y_k = \beta\left(b_k + \sum_{j=i}^{q} w_{jk}^2 h_j\right) \quad k = 1, 2, \ldots, r \tag{7.2.2}$$

where w_{jk}^2 is the weight from hidden layer node j to output layer node k, b_k is the bias of the node k in output layer, h_j is the output from node j of the hidden layer, and β is an activation function in output layer. The output value of the nodes in hidden layer is given by

$$h_j = \alpha\left(b_j + \sum_{z=1}^{p} w_{zj}^{1} i_z\right) \quad j = 1, 2, \ldots, q \tag{7.2.3}$$

where w_{zj}^{1} is the weight from input layer node z to hidden layer node j, b_j is the bias of the node j, i_z is the value in the input layer, and α is an activation function in hidden layer.

7.3 Neural Network Approaches in Software Reliability

A number of factors that normally demonstrate non-linear patterns such as software development methodology, software development environment, complexity of the software, software personnel, etc. affect the behavior of software reliability growth. This imposes several limitations on existing statistical modeling methods that depend highly on making assumptions on the testing process. Neural network models have a significant advantage over analytical models, because they require only failure history as input and no assumptions. Consequently, they have drawn attention of many researchers in recent years. It has been found that neural network methods can be applied to estimate the number of faults and predict the number of software failures as they often offered better results than existing statistical analytical models.

As reliability growth models exhibit different predictive capabilities at different testing phases both within a project and across projects, researchers are finding it nearly impossible to develop a universal model that will provide accurate predictions under all circumstances. A possible solution is to develop models that do not require making assumptions about either the development environment or external parameters. Recent advances in neural networks show that they can be used in applications that involve predictions. Neural network methods may handle numerous factors and approximate any non-linear continuous function.

Many papers are published in the literature addressing that neural networks offer promising approaches to software reliability estimation and prediction. Karunanithi and co-workers [15–18] first applied some kinds of neural network architecture to estimate the software reliability and used the execution time as input, the cumulative number of detected faults as desired output and encoded the input and output into the binary bit string. Furthermore, they also illustrated the usefulness of connectionist models for software reliability growth prediction and showed that the connectionist approach is capable of developing models of varying complexity. Khoshgoftaar and co-workers [19, 20] used the neural network as a tool for predicting the number of faults in programs. They introduced an approach for static reliability modeling and concluded that the neural networks produce models with better quality of fit and predictive quality.

Sherer [21] applied neural networks for predicting software faults in several NASA projects. Khoshgoftaar et al. [22] used the neural network as a tool for

predicting the number of faults in a program and concluded that the neural networks produce models with better quality of fit and predictive quality. Sitte [23] compared the predictive performance of two different methods of software reliability prediction: "neural networks" and "recalibration for parametric models." Cai et al. [24] used the recent 50 inter-failure times as the multiple-delayed-inputs to predict the next failure time and found the effect of the number of input neurons, the number of neurons in the hidden layer and the number of hidden layers by independently varying the network architecture. They advocated the development of fuzzy software reliability growth models in place of probabilistic software reliability models.

Most of the neural networks used for software reliability modeling can be classified into two classes. One used cumulative execution time as inputs and the corresponding accumulated failures as desired outputs. This class focuses on modeling software reliability modeling by varying different kind of neural network such as recurrent neural network [16]; Elman network [23]. The other class, models the software reliability based on multiple-delayed input single-output neural network. Cai and co-workers [24, 25] used the recent 50 inter-failure times as the multiple-delayed-inputs to predict the next failure time.

There was a common problem with all these of approaches. We have to predetermine the network architecture such as the number of neurons in each layer and the numbers of the layers. In Cai's experiment, he found the effect of the number of input neurons, the number of neurons in hidden layer and the number of hidden layers by independently varying the network architecture. Another problem is that since several fast training algorithms are investigated for reducing the training time, these advanced algorithms focus on the model fitting and this will cause the over fitting. When the network is trained, the error of training set is small for training data, but when new data is available to the network, the error maybe extremely large. Since the modeling approaches mentioned above treat the neural network as a black box, researchers consider the combinations of the network architecture to find a solution that can be suggested for us to build a network that can perform more accurate prediction. But we still cannot know about the meaning of each element of the network. Su et al. [14] proposed a neural-network-based approach to software reliability assessment. Unlike the traditional neural network modeling approach, they first explain the network networks from the mathematical viewpoints of software reliability modeling and then derived some very useful mathematic expressions that can directly applied to neural networks from traditional SRGM. They also showed how to apply neural network to software reliability modeling by designing different elements of the network architecture. The proposed neural-network-based approach can also combine various existing models into a dynamic weighted combinational model (DWCM). Kapur et al. [26] proposed an ANN based Dynamic Integrated Model (DIM), which is an improvement over DWCM given by Su et al. [14]. In another research Kapur et al. [27] have proposed generalized dynamic integrated ANN models for the existing fault complexity models. Khatri et al. [28] have proposed ANN based SRGM considering testing efficiency.

7.3.1 Building ANN for Existing Analytical SRGM

The objective function y of the neural network can be considered as compound function. By deriving a compound function from the conventional statistical SRGM, we can build a neural network based SRGM having all the properties of the existing SRGM. Simple feed forward neural network architecture for basic SRGM can be consisting of one hidden layer and one neuron each in input, hidden and output layer as shown in Fig. 7.3.

Neural network approach for software reliability measurement in general is based on building a network of neurons with weights. These weights have some initial value which is changed during training using back-propagation method to minimize the mean squared error. For each training sample, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual value. These modifications are made in the backward direction, that is, from the output layer, through each hidden layer down to the first hidden layer.

Network design is a trial-and-error process and may affect the accuracy of the resulting trained network. There are no clear rules to the best number of hidden layer units. The initial values of the weights may also affect the resulting accuracy. Once a network has been trained and its accuracy is not considered acceptable, it is common to repeat the training process with a different network topology or a different set of initial weights.

Cai et al. [24] had depicted some relationships between the neural network and conventional NHPP models as follows:

- $\beta(t)$ is equivalent to the mean value function of SRGM.
- w_1 is the failure rate.
- w_2 is the proportion of expected total number of faults latent in the system.

The output of the hidden layer, $\alpha(x)$, is similar to the distribution function.

For example, if we construct a neural network with an activation function $\alpha(i) = 1 - e^{-i}$ in the hidden layer, a pure linear activation function $\beta(i) = i$ in the output layer and no bias in hidden as well as in output layer, i.e. the input to the hidden layer with weight w_1 is

$$x(t) = w_1 t + b_1 \tag{7.3.1}$$

where b_1 is the bias, then the output of the hidden layer is given by

$$h(t) = \alpha(x(t)) = 1 - e^{-x(t)} \tag{7.3.2}$$

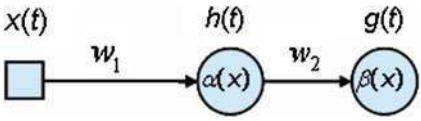


Fig. 7.3 Feed-forward network with single neuron in each layer

If the bias b_1 is negligible such that it can assumed to be zero then the output from the hidden layer is

$$h(t) = 1 - e^{-w_1 t} \tag{7.3.3}$$

Now the input to the output layer is

$$y(t) = h(t)w_2 + b_0 \tag{7.3.4}$$

If the bias b_0 is negligible such that it can be assumed to be zero then the output from the output layer is

$$g(t) = \beta(y(t)) = w_2(1 - e^{-w_1 t}) \tag{7.3.5}$$

If we assume $w_1 = b$ and $w_2 = a$ then Eq. (7.3.5) corresponds to the conventional Goel and Okumoto [29] model.

Back propagation algorithm used to train the network requires that the activation functions should be continuous and differentiable everywhere. The activation functions we have used above are continuous and differentiable everywhere. The parameters of the models are estimated based on the data. For training the ANN we use back propagation method. Kapur et al. [26] have written a software program written in C programming language for training of the ANN. The program can be modified according to the activation function used and the network architecture. The program requires a failure data set as input and generates estimates of the parameters of the network models as output. Using these estimated values reliability measurements are made.

7.3.2 Software Failure Data

Software failure is an incorrect result with respect to the specification or an unexpected software behavior perceived by the user, while software fault is the identified or hypothesized cause of the software failure. When a time basis is determined, failures can be expressed in the form of cumulative failure function. The cumulative failure function (also called the mean value function) denotes the average cumulative failures associated with each point of time. In software failure process, $m(t_i)$ is the cumulative number of faults removed by execution time t_i. Software reliability data are arranged in pairs as $\{t_i, m(t_i)\}$. Each pair of software reliability data is passed on to the neural network to determine the weights and then the trained ANN is used for estimating the cumulative failures in software at time t_i or predicting the cumulative failures at any future time.

7.3.2.1 Normalization

Software reliability data are normalized before applying on the neural-network. Normalization is performed by scaling the value of the collected data within a

small-specified range of 0.0–1.0. There are many methods for normalization such as *min–maxnormalization*, *z-scorenormalization* and *normalization by decimal scaling*. In this paper, we have used min–max normalization, which performs a linear transformation on the original data. For a data variable x having its minimum and maximum value $\min_x$ and $\max_x$, respectively, min–max normalization maps a data value_x to new_value_x in the range [new_min_x, new_max_x] using the formula

$$\text{new\_value}_x = \frac{\text{value}_x - \min_x}{\max_x - \min_x}(\text{new\_max}_x - \text{new\_min}_x) + \text{new\_min}_x \qquad (7.3.6)$$

Min–max normalization preserves the relationships among the original data values.

7.4 Neural Network Based Software Reliability Growth Model

7.4.1 Dynamic Weighted Combinational Model

Dynamic Weighted Combinational Model (DWCM) proposed by Su et al. [14] is a neural-network-based approach to software reliability assessment by combining various existing models. The design methodology of the network elements is described in detail by the authors. The model considers feedforward neural network architecture with single neuron in each of the input and output layers and three neurons in the hidden layer. Each of the neurons in the hidden layer receives a weighted input form the single input neuron. The output from each of the neurons in the hidden layer is weighted in different proportions and the combined weighted output (so the name of the model) of the hidden layer is fed as the input to the output layer which then determines the output of the network based on the activation function in the output layer. The activation function in each of the three neurons in the hidden layer is defined according to the Goel and Okumoto [29], Yamada et al. [30] delayed s-shaped and logistic growth curve models [31]. The neural network architecture for this scenario is depicted in Fig. 7.4.

The activation function for the three neurons in the hidden layers is given by Eqs. (7.4.1)–(7.4.3), respectively.

The jth neuron in hidden layer will have the activation function $\alpha_j(x)$. The activation functions for the units in hidden layer in Fig. 7.4 are defined as

$$\alpha_1(x) = 1 - \mathrm{e}^{-x} \qquad (7.4.1)$$

$$\alpha_2(x) = 1 - (1 + x)\mathrm{e}^{-x} \qquad (7.4.2)$$

$$\alpha_3(x) = \frac{1}{1 + \mathrm{e}^{-x}} \qquad (7.4.3)$$

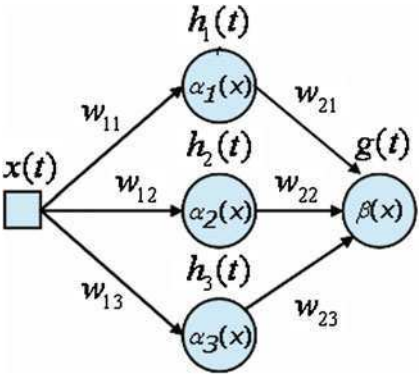


Fig. 7.4 Network architecture of the dynamic weighted combinational model

The activation function for the neuron in output layer is defined as

$$\beta(x) = x \tag{7.4.4}$$

w_{1j}, w_{2j} ($j = 1$, 2 and 3) are the weights assigned in the network form input layer to hidden layer and hidden layer to output layer, respectively. This network architecture assumes no bias in the neurons of hidden layer and output layer.

Input to the first, second and third neurons of hidden layer are, respectively,

$$x_1(t) = w_{11}t \tag{7.4.5}$$

$$x_2(t) = w_{12}t \tag{7.4.6}$$

$$x_3(t) = w_{13}t \tag{7.4.7}$$

Corresponding to the above inputs, outputs from each of the neurons in hidden layers is, respectively,

$$h_1(t) = \alpha_1(w_{11}t) = 1 - \mathrm{e}^{-w_{11}t} \tag{7.4.8}$$

$$h_2(t) = \alpha_2(w_{12}t) = 1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t} \tag{7.4.9}$$

$$h_3(t) = \alpha_3(w_{13}t) = \frac{1}{1 + \mathrm{e}^{-(w_{13}t)}} \tag{7.4.10}$$

Input to the single unit of output layer is

$$y(t) = \beta\left(w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t}) + w_{23}\left(\frac{1}{1 + \mathrm{e}^{-(w_{13}t)}}\right)\right) \tag{7.4.11}$$

hence the output from the single unit of output layer is

$$g(t) = w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t}) + w_{23}\left(\frac{1}{1 + \mathrm{e}^{-(w_{13}t)}}\right) \tag{7.4.12}$$

Note that if we assume w_{1j} ($j = 1, 2$ and 3) are equivalent to the fault detection rates of the conventional SRGM and w_{2j} ($j = 1, 2$ and 3) are equivalent to the proportion of total fault content in the software, i.e., $w_{11} = b_1$, $w_{21} = a_1$, $w_{12} = b_2$, $w_{22} = a_2$, $w_{13} = b_3$ and $w_{23} = a_3$, Eq. (7.4.12) can be written as

$$m(t) = a_1\left(1 - e^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)e^{-b_2 t}\right) + \frac{a_3}{1 + e^{-b_3 t}} \tag{7.4.13}$$

Also note that w_{2j} are the weights of each individual model and their values can be determined by the training algorithm. Actually application of models in practice becomes more effective by combining them. The approach can automatically determine the weight of each model according to the characteristics of the selected data sets.

The above discussion describes in detail how a neural network is constructed for the selected conventional SRGM. Prediction of the software reliability using the neural network approach consists of the following sequential steps.

1. Select some appropriate (and suitable) SRGM (at least one).
2. Construct the neural network of selected models by designing the activation functions and bias.
3. Gather the data set of the software failure history. Normalize the cumulative execution time t_i and compute its corresponding accumulated number of software failure m_i
4. Feed all pairs of $\{t_i, m_i\}$ to the network and train the network by using the back-propagation algorithm.
5. When the network is trained, feed the future testing time to the network, and the network output is the possible cumulative number of software failures in the future.

The activation functions for the three neurons in the hidden layers in the above model are defined corresponding to Goel and Okumoto [29], Yamada et al. [30] and logistic growth curve, respectively. The logistic growth curve is quite often used in statistical literature for describing the growth of the various types of the phenomena such as population growth and in general gives good results in many cases. The literature of software reliability also support the use of logistic function for the reliability growth modeling however in software reliability modeling it is more often called a learning curve as it can be modified to capture the learning phenomena which is of most common occurrence in most of the testing process. Kapur and Garg [32] SRGM is among the earliest and most commonly used SRGM that address to the learning phenomena in the testing process. The logistic growth curve in the DWCM can be replaced by this learning curve if we do not ignore the bias in the third neuron of the hidden layer while again ignoring the bias in the output layer [26]. In that case the above model is redefined as follows.

If we do not ignore the bias in the third neuron of the hidden layer then the input to this neuron will be

$$x_3(t) = w_{13} t + c \tag{7.4.14}$$

where c is the bias. Corresponding to the above input, output from this neurons in hidden layer is

$$h_3(t) = \alpha_3(w_{13}t + c) = \frac{1}{1 + \mathrm{e}^{-(w_{13}t+c)}} = \frac{1}{1 + \gamma\,\mathrm{e}^{-(w_{13}t)}} \tag{7.4.15}$$

where $\gamma = \mathrm{e}^{-c}$. In this case the combined input to the single unit of output layer is

$$y(t) = \beta\left(w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t}) + w_{23}\left(\frac{1}{1 + \gamma\,\mathrm{e}^{-(w_{13}t)}}\right)\right) \tag{7.4.16}$$

hence the output from the single unit of output layer is

$$g(t) = w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t}) + w_{23}\left(\frac{1}{1 + \gamma\,\mathrm{e}^{-(w_{13}t)}}\right) \tag{7.4.17}$$

In this way considering the substantial bias in the third neuron of the hidden layer the ANN can incorporate the learning nature of the testing process. Now similar to (7.4.13), Eq. (7.4.17) can be written as

$$m(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) + \frac{a_3}{1 + \gamma \mathrm{e}^{-b_3 t}} \tag{7.4.18}$$

The above ANN model is the dynamic weighted combinational ANN model for the exponential [29], s-shaped [30] and flexible [32] learning model. As this model combined all the three types exponential, s-shaped and flexible model into a single model it provides a good fit for a number of real life applications.

7.4.2 Generalized Dynamic Integrated SRGM

The NHPP based software reliability growth models discussed throughout the book are either exponential or s-shaped in nature. In exponential software reliability growth models, software reliability growth is defined by the mathematical relationship that exists between the time span of using (or testing) a program and the cumulative number of errors discovered. In contrast, s-shaped software reliability growth is more often observed in real life projects. There are many reasons why observed software reliability growth curves often become s-shaped. S-Shaped software reliability growth curve is typically caused by the definition of failures. The growth is also caused by the continuous test effort increase in which the test effort has been incrementally increased through the test period. Some of these causative factors or influences can be described by making the basic assumptions of the exponential growth model more realistic.

A number of models discussed in the book refer to the complexity of faults in the software. These models define that any software can be assumed to contain n different types of faults, the type of fault is distinguished based on the delay time between their observation and removal. Application of fault complexity based models on real life projects in general produces good results as different types of faults are treated differently. The neural network approach for reliability estimation and prediction [14] can combine a number of SRGM with different weights. This idea generates the thought of developing a neural-network-based model that can in general combine n different SRGM, one for each type of fault.

This section presents an ANN-based Generalized Dynamic Integrated SRGM (GDIM) [27] that can be applied for reliability estimation for a software project expected to contain different (n) types of faults. Recall that the time delay between the failure observation and subsequent fault removal represents the complexity of the faults. More severe the fault more will be time delay. The faults are classified as *simple*, *hard* and *complex*. The fault is classified as *simple* if the time delay between failure observation, isolation and removal is negligible. If there is a time delay between failure observation and isolation, the fault is classified as a *hard* fault. If there is a time delay between failure observation, isolation and removal, the fault is classified as a *complex* fault. For detailed modeling refer to Sect. 2.4.

Assuming that the software consists of n different types of faults and on each type of fault, a different strategy is required to remove the cause of failures, Kapur et al. [33] assumed that for a type i ($i = 1, 2, \ldots, n$) fault, i different processes (stages) are required to remove the cause of the failure. We can apply the neural-network-based approaches to build a GDIM to predict and estimate the reliability of software consisting of n different types of faults depending upon their complexity. The neural network is constructed with single input, single output but with more than one neuron in the hidden layer. The number of units in the hidden layer depends on the types of faults in the software system. For software having n different types of faults on basis of their complexity, there will be n units in the hidden layer. Such a feedforward neural network is shown in Fig. 7.5.

In practice, we can design different activation functions on different units in the hidden layer. The jth neuron in the hidden layer will have the activation function for jth type of fault. The weights w_{1j}, from the input layer to the jth node in the hidden layer, represent the fault detection rate of jth type of fault and weights w_{2j}, from the jth node in the hidden layer to the single node in the output layer represent the proportionality of total number of jth type of faults latent in the system. There will be no bias in units of hidden layer and in the single unit of output layer. The activation function for the jth node in the hidden layer is defined as

$$\alpha_j(x) = 1 - \mathrm{e}^{-x} \sum_{i=0}^{j-1} \frac{(x)^i}{i!} \tag{7.4.19}$$

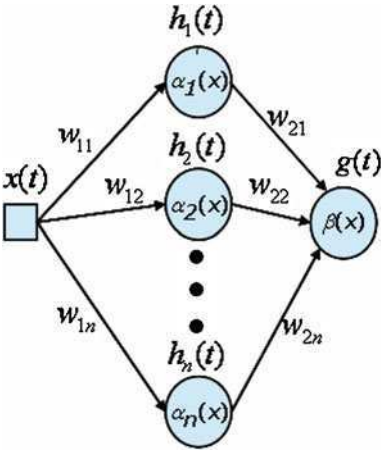


Fig. 7.5 Network Architecture of GDIM for n types of faults

While the activation function for the node in the output layer can be defined same as in (7.4.3). Proceeding in the similar manner as in case of Su et al. [14] model we can define the network output. w_{1j}, w_{2j} ($j = 1, 2,\ldots, n$) are the weights assigned in the network form input layer to hidden layer and hidden layer to output layer, respectively. If we assume no bias in both of hidden layer and output layer input to ith unit of hidden layer is given by

$$x_j(t) = w_{1j}t \tag{7.4.20}$$

The activation function $\alpha_j(x)$ determines the outputs from the neurons in hidden layers

$$h_j(t) = \alpha_j(w_{1j}t) = 1 - \mathrm{e}^{-x}\sum_{i=0}^{j-1}\frac{(w_{1j})^i}{i!} \tag{7.4.21}$$

The weighted output of the hidden layer is fed as input to the output layer, i.e.

$$y(t) = \beta\left(\sum_{j=1}^{n} w_{2j}\left(1 - \mathrm{e}^{-x}\sum_{i=0}^{j-1}\frac{(w_{1j})^i}{i!}\right)\right) \tag{7.4.22}$$

hence the output from the single unit of output layer is

$$g(t) = \sum_{j=1}^{n} w_{2j}\left(1 - \mathrm{e}^{-x}\sum_{i=0}^{j-1}\frac{(w_{1j})^i}{i!}\right) \tag{7.4.23}$$

In Eq. (7.4.23) if we replace the weights w_{1j} by b_j, $j = 1, 2,\ldots, n$ (fault detection rate for ith type of fault) $j = 1, 2,\ldots, n$ and weights w_{2j} by a_j, $j = 1, 2,\ldots, n$ (proportion of total fault content for ith type of fault) then the network output can be represented as

$$m(t) = \sum_{j=1}^{n} a_j \left(1 - \mathrm{e}^{-x} \sum_{i=0}^{j-1} \frac{(b_j)^i}{i!}\right) \tag{7.4.24}$$

A trained ANN of this type determines the weight of each network connection according to the characteristics of the selected data sets hence can determine the types of faults present in the software and their proportion in the total fault content.

An application

If the software contains two types of faults then the network architecture is GDIM as shown in (Fig. 7.6).

In this case the network output is

$$g(t) = w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t}) \tag{7.4.25}$$

This network output corresponds to the weighted sum of Goel and Okumoto [29] exponential SRGM for the simple faults and Yamada delayed s-shaped SRGM for the hard types of faults. The combined weighted mean value function of the SRGM corresponding to this network output is

$$m(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) \tag{7.4.26}$$

Similarly different ANN models can be designed for the different number of faults in the software according to their complexity.

The generalized dynamic integrated neural network model accounts for a very important aspect of software testing, i.e., fault complexity. However, as noted earlier learning is a major factor that effect the testing progress and a model that incorporates the learning factor provides good results in a number of cases. Besides that a learning model having flexible formulation which can describe both of the most common types of failure growth curves, i.e., exponential as well as s-shaped depending on the observed failure data characteristics turns out to be more useful. With this view Kapur et al. [34] further modified the neural network design of GDIM to incorporate the learning phenomenon so that the model can be

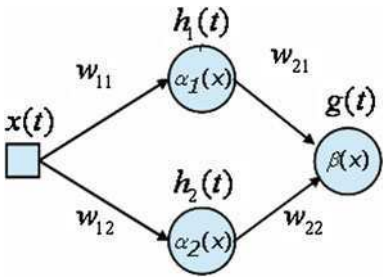


Fig. 7.6 Network architecture of GDIM with two types of faults

applied to a verity of failure data sets. To accommodate the impact of leaning in the generalized DIM catering to fault of different complexity the activation functions on the neurons of the hidden layer are redefined incorporating the learning parameters. If there is only one neuron in the hidden layer then the software is assumed to contain only one type of fault. If we assume this fault type to be simple then an exponential activation function can be chosen for this neuron. Hence for the first neuron we continue with the activation function given by

$$\alpha_1(x) = 1 - \mathrm{e}^{-x} \tag{7.4.27}$$

and for the other neurons the general functional form of the activation function in the hidden layer is

$$\alpha_j(x) = \frac{1 - \mathrm{e}^{-x+c_{j-1}} \sum_{i=0}^{j-1} (x - c_{j-1})^i / i!}{1 + \mathrm{e}^{-x}} \quad j = 2, \ldots, n \tag{7.4.28}$$

assuming a total of n neurons in the hidden layer. The activation function in the single neuron in the output layer will remain the same as in GDIM.

Proceeding in the similar manner with weights w_{1j}, w_{2j} ($j = 1,2, \ldots,n$) in the transformations from input layer to hidden layer and hidden layer to output layer, respectively. It is assumed that there will be no bias in first unit of hidden layer and in the single unit of the output layer. All other neurons in hidden layer has a bias c_{j-1}, i.e., c_1 is the bias in the second unit, c_2 is the bias in the third unit and so on in the hidden layers.

Input to the units of hidden layers is given by

$$x_1(t) = w_{11} t \tag{7.4.29}$$

$$x_j(t) = w_{1j} t + c_{j-1}, \quad j = 2, \ldots, n \tag{7.4.30}$$

The activation function $\alpha_j(x)$ determines the outputs from the neurons in hidden layers

$$\begin{aligned} h_j(t) = \alpha_j\big(x_j(t)\big) &= \begin{cases} 1 - \mathrm{e}^{-w_{11}}, & j = 1 \\ \dfrac{1 - \mathrm{e}^{-w_{1j}} \sum_{i=1}^{j-1} (w_{1j})^i / i!}{1 + \mathrm{e}^{-(w_{1j} + c_{j-1})}}, & j = 2, \ldots, n \end{cases} \\ &= \begin{cases} 1 - \mathrm{e}^{-w_{11}}, & j = 1 \\ \dfrac{1 - \mathrm{e}^{-w_{1j}} \sum_{i=1}^{j-1} (w_{1j})^i / i!}{1 + \gamma_j \mathrm{e}^{-w_{1j}}}, & j = 2, \ldots, n \end{cases} \end{aligned} \tag{7.4.31}$$

where $\gamma_j = c_{j-1}\ j = 2, \ldots, n$.

The weighted output of the hidden layer is fed as input to the output layer, i.e.

$$y(t) = \beta \left(w_{21}(1 - \mathrm{e}^{-w_{11}}) + \sum_{j=2}^{n} w_{2j} \left(\frac{1 - \mathrm{e}^{-w_{1j}} \sum_{i=1}^{j-1} (w_{1j})^i / i!}{1 + \gamma_j \mathrm{e}^{-w_{1j}}} \right) \right) \tag{7.4.32}$$

hence the output from the single unit of output layer is

$$g(t) = w_{21}(1 - \mathrm{e}^{-w_{11}}) + \sum_{j=2}^{n} w_{2j}\left(\frac{1 - \mathrm{e}^{-w_{1j}} \sum_{i=1}^{j-1} (w_{1j})^i / i!}{1 + \gamma_j \mathrm{e}^{-w_{1j}}}\right) \tag{7.4.33}$$

Assuming weights $w_{1j} = b_j$, and $w_{2j} = a_i$, $j = 1,\ldots, n$ the network output can be represented as

$$m(t) = a_1\left(1 - \mathrm{e}^{-b_1}\right) + \sum_{j=2}^{n} a_j\left(\frac{1 - \mathrm{e}^{-b_j} \sum_{i=1}^{j-1} (b_j)^i / i!}{1 + \gamma_j \mathrm{e}^{-b_j}}\right) \tag{7.4.34}$$

An application

We now demonstrate neural network architecture for the flexible GDIM for software expected to contain three types of faults. The pictorial architecture of such a network is shown in Fig. 7.4. This type of neural network is having one neuron in the input and the output layers and three neurons in the single hidden layer.

The activation functions in the three neurons of the hidden layer will be

$$\alpha_1(x) = 1 - \mathrm{e}^{-x} \tag{7.4.35}$$

$$\alpha_2(x) = \frac{1 - (1 + x - c_1)\mathrm{e}^{-x+c_1}}{1 + \mathrm{e}^{-x}} \tag{7.4.36}$$

$$\alpha_3(x) = \frac{1 - \left(1 + (x - c_2) + ((x - c_2)^2/2)\right)\mathrm{e}^{-x+c_2}}{1 + \mathrm{e}^{-x}} \tag{7.4.37}$$

With these activation functions and inputs from the hidden layers according to (7.4.29) and (7.4.30) and following the similar procedure as above the network output will be given by

$$\begin{aligned} g(t) = {} & w_{21}(1 - \mathrm{e}^{-w_{11}t}) + w_{22}\frac{(1 - (1 + w_{12}t)\mathrm{e}^{-w_{12}t})}{1 + \mathrm{e}^{-w_{12}t - c_1}} \\ & + w_{23}\frac{\left(1 - \left(1 + w_{13}t + \frac{(w_{13}t)^2}{2}\right)\mathrm{e}^{-w_{13}t}\right)}{1 + \mathrm{e}^{-w_{13}t - c_1}} \end{aligned} \tag{7.4.38}$$

this network output correspond to the weighted sum of Goel and Okumoto [29] exponential SRGM for the simple faults and flexible delayed s-shaped SRGM for the hard types of faults and flexible Erlang 3-stage SRGM for the complex faults. The combined weighted mean value function of the SRGM corresponding to this network output is

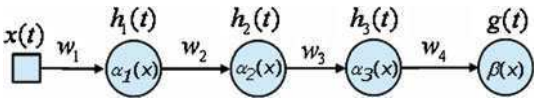


Fig. 7.7 Neural network architecture for testing efficiency based SRGM

$$m(t) = a_1\left(1 - e^{-b_1 t}\right) + \frac{a_2\left(1 - (1 + b_2 t)e^{-b_2 t}\right)}{1 + \gamma_2 e^{-b_2 t}} + a_3 \frac{\left(1 - \left(1 + b_3 t + \left((b_3 t)^2/2\right)\right)e^{-b_3 t}\right)}{1 + \gamma_3 e^{-b_3 t}} \tag{7.4.39}$$

7.4.3 Testing Efficiency Based Neural Network Architecture

Neural network architecture for a testing efficiency based SRGM [28] consist of multiple hidden layers. ANN architecture for a software reliability growth model that can address to the two types of imperfect debugging can be designed considering three hidden layers each with a single neuron. Such a neural network is represented pictorially in Fig. 7.7.

Each of the activation function on each of the neuron in the hidden layer may or may not have similar activation function. The SRGM we address to in this section considers different activation functions on each of the neuron. The activation function $\alpha_j(x)$ on the jth hidden layer h_j is defined as follows

$$\alpha_1(x) = x \tag{7.4.40}$$

$$\alpha_2(x) = x \tag{7.4.41}$$

$$\alpha_3(x) = 1 - e^{-x} \tag{7.4.42}$$

The activation function for the output layer is defined as

$$\beta(x) = \frac{x}{1 - \alpha} \tag{7.4.43}$$

If w_1 is the weight from the input layer to first hidden layer, w_2 is the weight from the first hidden layer to second hidden layer, w_3 is the weight from the second hidden layer to third hidden layer and w_4 is the weight from the third hidden layer to output layer and there is no bias in any of the transformations then mathematically the network architecture is defined as:

Input to the first hidden layer is

$$x_1(t) = w_1 t$$

output of the first hidden layer is

$$h_1(t) = \alpha_1(x_1(t)) = \alpha_1(w_1 t) = w_1 t$$

second hidden layer receives the input

$$x_2(t) = w_2 w_1 t$$

from the first hidden layer and generates the output

$$h_2(t) = \alpha_2(x_2(t)) = \alpha_2(w_2 w_1 t) = w_2 w_1 t$$

the input that goes to the third hidden layer is

$$x_3(t) = w_3 w_2 w_1 t$$

and the output generated from the third hidden layer is

$$h_3(t) = \alpha_2(x_3(t)) = \alpha_3(w_3 w_2 w_1 t) = 1 - \mathrm{e}^{-w_3 w_2 w_1 t}$$

The input to the output layer is

$$y(t) = w_4(1 - \mathrm{e}^{-w_3 w_2 w_1 t})$$

Output from the output layer is

$$g(t) \equiv m(t) = \beta(y(t)) = \frac{w_4}{(1-\alpha)}(1 - e^{-w_3 w_2 w_1 t})$$

if we assume $w_4 = a, w_3 = b, w_2 = p$ and $w_1 = (1-\alpha)$ then it represents the mean value function for SRGM incorporating two types of imperfect, i.e.

$$g(t) \equiv m(t) = \frac{a}{(1-\alpha)}(1 - e^{-bp(1-\alpha)t}). \tag{7.4.44}$$

7.5 Data Analysis and Parameter Estimation

Neural network approach for software reliability assessment is based on building a network of units with initialized weights which are changed during training using training algorithm to minimize the mean squared error. We choose the back propagation algorithm to train the network. Back-propagation algorithm trains the neural network by minimizing the squares of the distance between the network output value and the corresponding desired output value. The method is preferred as its formulas are identical to the method of least squares which minimizes the sum of squares of the distance between the best fit line and the actual data points with identical formulas. However there is a lot of difference between the two methods but due to the identical approach is chosen here for the performance analysis of the NN models discussed in this chapter. Su and Huang used the NN methods proposed by Karunanithi and Malaiya [18] and Tian and Noore [35] for the purpose. While

Kapur et al. [26] programmed the method in C programming language which can be modified easily according to the number of hidden layers and the number of neurons in each hidden layer. The program requires approximation for the initial weights, based on them train the network and determines the network output. One can also use the neural network module available in many software such as SPSS, Matlab, Mathematica, etc. for the purpose. The results of our analysis are based on Kapur et al. [26] approach. Time and Cumulative failure information is normalized between 0 and 1 before passing to neural network architecture.

Failure Data Set

Most of the NN models are weighted combinational models and can be used to represent the fault of different complexity. Therefore, the data set which illustrates different types of faults in the software is taken for the analysis. The interval domain data is taken from Misra [36] in which the number of faults detected per week (38 weeks) is specified and a total of 231 failures are observed. The data describes that three types of faults—minor (64.07%), major (34.2%) and critical (1.73%) are present in the software. Mean square of error (MSE) and root mean square prediction error (RMSPE) are taken as the goodness of fit criteria.

Following neural network models have been chosen for data analysis and parameter estimation.

Model 1 (M1): DWCM [14]

$$m(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) + \frac{a_3}{1 + \mathrm{e}^{-b_3 t}}$$

Model 2 (M2): DWCM [26]

$$m(t) = a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) + \frac{a_3}{1 + \gamma\mathrm{e}^{-b_3 t}}$$

Model 3 (M3): GDIM [27], corresponding to three neurons in the hidden layer.

$$\begin{aligned} m(t) = {} & a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right) \\ & + a_3\left(1 - \left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)\mathrm{e}^{-b_3 t}\right) \end{aligned}$$

Model 4 (M4): Flexible GDIM [34], corresponding to three neurons in the hidden layer.

$$\begin{aligned} m(t) = {} & a_1\left(1 - \mathrm{e}^{-b_1 t}\right) + \frac{a_2\left(1 - (1 + b_2 t)\mathrm{e}^{-b_2 t}\right)}{1 + \gamma_2\mathrm{e}^{-b_2 t}} \\ & + a_3\frac{\left(1 - \left(1 + b_3 t + \left((b_3 t)^2/2\right)\right)\mathrm{e}^{-b_3 t}\right)}{1 + \gamma_3\mathrm{e}^{-b_3 t}} \end{aligned}$$

Model 5 (M5): Testing efficiency model [28]

Table 7.1 Estimation result for models 1–5

Model	Estimated parameters								Comparison criteria	
	a, a_1	a_2	a_3	b, b_1	b_2	b_3	γ, γ_1, p	γ_2, α	MSE	RMSPE
M1	228	203	20	0.0270	0.0290	0.0150	–	–	15.35	3.64
M2	367	11	72	0.0120	0.0050	0.0310	10.85	–	18.12	4.15
M3	144	4	303	0.0860	0.0010	0.0660	–	–	18.64	4.03
M4	106	116	228	0.1400	0.0640	0.0970	228	0.10	18.30	3.85
M5	466	–	–	0.0170	–	–	0.9530	0.0190	27.44	5.65

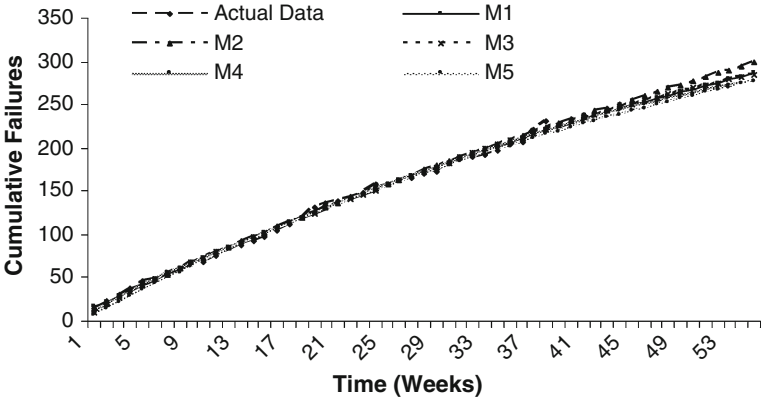


Fig. 7.8 Goodness of fit curve for models M1–M5

$$m(t) = \frac{a}{(1-\alpha)}\left(1 - \mathrm{e}^{-bp(1-\alpha)t}\right)$$

The values of estimated parameters have been tabulated in Table 7.1. Figure 7.8 shows the goodness of fit curves for the estimation results and future predictions.

Model M1–M4 all define faults of different complexity and have given very close result on the data. However, the model M1 yields the best fit curve with lowest values of MSE (15.35) and RMSPE (3.64). The testing efficiency model does not give a good fit on the data. The proportions of faults of type simple, hard and complex according to the model M1 is $p_1 = 50.5\%$, $p_2 = 45\%$ and $p_3 = 4.5\%$, respectively, which also seems to be following the actual data sets. The result of models M3 and M4 shows a very high proportion of critical faults, although the actual data reflects a very less proportion of critical faults. Consider the case of M3 according to which the a total of 144 simple faults are present in the software, however the actual data specifies that in 38 period of testing 148 simple faults have already been removed. This is contradictory. Also the estimated figure of hard fault content according to model M2 is very low in contrast to the actual data. All this suggest that model M1 adequately describe this data set.

We have discussed several neural network based models for software reliability estimation. However, the research is this area is still in infancy stage. Lot more idea generation and application is required. Apart from this the area also demands to develop new methods and algorithms to train the network.

Exercises

1. Explain the basic structure of an artificial neural network.
2. What is the difference between a feed-forward and a feedback neural network?
3. Explain the back propagation learning algorithm for feed forward neural network.
4. Explain how we can build an ANN for the exponential SRGM given by the equation.

$$m(t) = a\left(1 - e^{-bt}\right)$$

5. Describe the structure of an ANN to describe the failure process of software containing four types of fault, giving mathematically inputs and outputs from the layers of the network.
6. Using the data from Sect. 7.5 estimate the model parameters of the SRGM developed in exercise 5. Compute the mean square error of estimation.

References

1. Stergiou C, Siganos D (1996) Neural networks. www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html
2. McCulloch WS, Pitts WH (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5:115–133
3. Farley BG, Clark WA (1954) Simulation of self-organizing systems by digital computers. Trans IRE PGIT 4:76–84
4. Rochester N, Holland JH, Habit LH, Duda WL (1956) Tests on a cell assembly theory of the action of the brain, using a large digital computer. IRE Trans Info Theory IT-2:80–93
5. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the Brain, Cornell Aeronautical Laboratory. Psychol Rev 65(6):386–408. doi: 10.1037/h0042519
6. Widrow B, Hoff ME Jr (1960) Adaptive switching circuits. IRE WESCON Conv Rec 4:96–104
7. Minsky ML, Papert SA (1969) Perceptrons: an introduction to computational geometry, expanded edition. MIT Press, Cambridge
8. Carpenter GA, Grossberg S (1988) The ART of adaptive pattern recognition by a self-organizing neural network. Computer 21(3):77–88
9. Klopf AH (1972) Brain function and adaptive systems—a heterostatic theory. Air Force Cambridge Research Laboratories Research, Bedford
10. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D., Harvard University, Cambridge

11. Anderson JA (1977) Neural models with cognitive implications. In: LaBerge D, Samuels SJ (eds) Basic processes in reading perception and comprehension. Erlbaum, Hillsdale, pp 27–90
12. Kohonen T (1977) Associative memory: a system theoretical approach. Springer, New York
13. Amari S (1967) A theory of adaptive pattern classifiers. IEEE Trans Electron Comput 16(3):299–307
14. Su YS, Huang CY (2007) Neural network based approaches for software reliability estimation using dynamic weighted combinational models. J Syst Softw 80:606–615
15. Karunanithi N, Malaiya YK, Whitley D (1991) Prediction of software reliability using neural networks. In: Proceedings 2nd IEEE international symposium on software reliability engineering, Los Alamitos, CA, pp 124–130
16. Karunanithi N, Whitley D, Malaiya YK (1992) Using neural networks in reliability prediction. IEEE Softw 9:53–59
17. Karunanithi N, Malaiya YK (1992) The scaling problem in neural networks for software reliability prediction. In: Proceedings 3rd international IEEE symposium of software reliability engineering, Los Alamitos, CA, pp 76–82
18. Karunanithi N, Malaiya YK (1996) Neural networks for software reliability engineering. In: Lyu MR (ed) Handbook of software reliability engineering. McGraw-Hill, New York, pp 699–728
19. Khoshgoftaar TM, Pandya AS, More HB (1992) A neural network approach for predicting software development faults. In: Proceedings 3rd IEEE international symposium on software reliability engineering, Los Alamitos, CA, pp 83–89
20. Khoshgoftaar TM, Szabo RM (1996) Using neural networks to predict software faults during testing. IEEE Trans Reliab 45(3):456–462
21. Sherer SA (1995) Software fault prediction. J Syst Softw 29(2):97–105
22. Khoshgoftaar TM, Allen EB, Hudepohl JP, Aud SJ (1997) Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Trans Neural Netw 8(4):902–909
23. Sitte R (1999) Comparison of software reliability growth predictions: neural networks vs. parametric recalibration. IEEE Trans Reliab 48(3):285–291
24. Cai KY, Cai L, Wang WD, Yu ZY, Zhang D (2001) On the neural network approach in software reliability modeling. J Syst Softw 58(1):47–62
25. Cai KY (1998) Software defect and operational profile modeling. Kluwer Academic Publishers, Dordrecht
26. Kapur PK, Khatri SK, Yadav K (2008) An artificial neural-network based approach for developing a dynamic integrated software reliability growth model. Presented in international conference on present practices and future trends in quality and reliability, ICONQR08, 22–25 Jan 2008
27. Kapur PK, Khatri SK, Goswami DN (2008) A generalized dynamic integrated software reliability growth model based on artificial neural network approach. In: Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems. Macmillan advanced research series. MacMillan India Ltd, New Delhi, pp 813–838
28. Khatri SK, Kapur R, Sehgal VK (2008) Neural-network based software reliability growth modeling with two types of imperfect debugging. Presented in the 40th Annual Convention of ORSI, 4–6 Dec 2008
29. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R-28(3):206–211
30. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R-32(5):475–484
31. Su YS, Huang CY, Chen YS (2005) An artificial neural-network based approach to software reliability assessment. In: CD-ROM proceedings 2005 IEEE region 10 conference (TENCON 2005), Nov. 2005, Melbourne, Australia, EI
32. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294

33. Kapur PK, Younes S, Agarwala S (1995) Generalized Erlang software reliability growth model. ASOR Bull 35(2):273–278
34. Kapur PK, Khatri SK, Basirzadeh M (2008) Software reliability assessment using artificial neural network based flexible model incorporating faults of different complexity. Int J Reliab Qual Saf Eng 15(2):113–127
35. Tian L, Noore A (2005) Evolutionary neural network modeling for software cumulative failure time prediction. Reliab Eng Syst Saf 87:45–51
36. Misra PN (1983) Software reliability analysis. IBM Syst J 22:262–270

Chapter 8
SRGM Using SDE

8.1 Introduction

A number of NHPP based SRGM have been discussed in the previous chapters. These models treat the event of software fault detection/removal in the testing and operational phase as a counting process in discrete state space. If the size of software system is large, the number of software faults detected during the testing phase becomes large, and the change in the number of faults, which are detected and removed through debugging activities, becomes sufficiently small compared with the initial fault content at the beginning of the testing phase. Therefore, in such a situation, the software fault detection process can be well described by a stochastic process with continuous state space. This chapter focuses on the development of stochastic differential equation based software reliability growth models to describe the stochastic process with continuous state space. Before developing any model we introduce the readers with the theoretical and mathematical background of stochastic differential equations.

8.2 Introduction to Stochastic Differential Equations

8.2.1 Stochastic Process

A stochastic process $\{X(t), t \in T\}$ is a collection of random variables, i.e., for each $t \in T$,$X(t)$ is a random variable. The index t is interpreted as time and as a result, we refer to $X(t)$ as the state of the process at time t.

The set T is called the index set of the process. When T is a countable set, the stochastic process is said to be a discrete time process. If T is an interval of real time, the stochastic process is said to be a continuous time process. For instance, $\{X_n, n = 0, 1, \ldots\}$ is a discrete time stochastic process indexed by the

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_8,
© Springer-Verlag London Limited 2011

non-negative integers, while $\{X(t),\ t \geq 0\}$ is a continuous time stochastic process indexed by the non-negative real numbers.

8.2.2 Stochastic Analog of a Classical Differential Equation

If we allow for some randomness in some of the coefficients of a differential equation, we often obtain a more realistic mathematical model of the situation.

Consider the simple population growth model

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = a(t)N(t) \tag{8.2.1}$$

where

$$N(0) = N_0 \text{ (a constant)}$$

and $N(t)$ is the size of the population at time t and $a(t)$ is the relative rate of growth at time t. It might happen that $a(t)$ is not completely known, but subject to some random environment effects, so that we have

$$a(t) = r(t) + \text{“noise”} \tag{8.2.2}$$

We do not know the exact behavior of the noise term, only its probability distribution and the function $r(t)$ is assumed to be a non-random.

8.2.3 Solution of a Stochastic Differential Equation

Stating the problem based on stochastic differential equations, we now explain the method to solve the problem.

The mathematical model for a random quantity is a random variable. A stochastic process is a parameterized collection of random variables $\{X(t), t \in T\}$ defined on a probability space (Ω, F, p) and assuming values in R^n.

8.2.3.1 σ-Algebra

If Ω is a given set, then a σ-algebra F on Ω is a family F of subset of Ω with the following properties:

1. $\varphi \in F$
2. $F_1 \notin F \Rightarrow F_1^C \Rightarrow F$, where F_1^C is complement of F in Ω.
3. $A_1, A_2, \ldots \in F \Rightarrow A = \bigcup_{i=1}^{\infty} A_i \in F$

The pair (Ω, F) is called a measurable space.

8.2.3.2 Probability Measure

A probability measure p on a measurable space (Ω, F) is function p: $F \to [0, 1]$ such that

1. $p(\Phi) = 0, p(\Omega) = 1$
2. if $A_1, A_2, \ldots \in F$ and $\{A_i\}_{i=1}^{\infty}$ is disjoint, i.e., $(A_i \wedge A_j = \Phi \text{ if } i \neq j)$ then $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$

8.2.3.3 Probability Space

The triplet (Ω, F, P) is called the probability space.

8.2.3.4 Brownian Motion

In 1828 the Scottish botanist Robert Brown observed that pollen grains suspended in liquid perform irregular motions. He and others noted that the path of a given particle is very irregular, having a tangent at no point, and the motion of two distinct particles appears to be independent. The motion was later explained by the random collision with the molecules of the liquid. To describe the motion mathematically it is natural to use the concept of a stochastic process $W(t)$, interpreted as the position at time t.

Definition A real-valued stochastic process $W(\cdot)$ is called a Brownian or Wiener process if

1. $W(0) = 0$
2. $W(t) - W(s)$ is $N(0, t - s)$ for $t \geq s \geq 0$
3. for times $0 < t_1 < t_2 < \cdots < t_n$ the random variables $W(t_1)$, $W(t_2) - W(t_1)$, …, $W(t_n) - W(t_{n-1})$ are independent.

In particular

$$p[W(0) = 0] = 1$$
$$E[W(t)] = 0,$$
$$E\left[W^2(t)\right] = t \text{ for real time } t \geq 0 \cdot \text{i.e. } \mathrm{Var}(W(t)) = t$$

and

$$E[W(t)W(t')] = \mathrm{Min}[t, t']$$

$W(t)$ follows normal distribution with mean 0 and variance t, so for all $t > 0$ and $a \leq b$ we have

$$p[a \le W(t) \le b] = \frac{1}{\sqrt{2\pi t}} \int_a^b \mathrm{e}^{-1/2(W^2(t))}\, \mathrm{d}w(t) \tag{8.2.3}$$

8.2.3.5 ItÔ Integrals [1, 2]

We now turn to the question of finding a reasonable mathematical interpretation of the "*noise*" term in Eq. (8.2.2)

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = N(t)(r(t) + \text{“noise”}) \tag{8.2.4}$$

or more generally in equation of the form

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = b(t, N(t)) + \sigma(t, N(t)) \cdot \text{noise} \tag{8.2.5}$$

where b and σ are some given functions.

Let us first concentrate on the case where the noise is one-dimensional. It is reasonable to look for some stochastic process $\gamma(t)$ to represent the noise term, so that

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = b(t, N(t)) + \sigma(t, N(t))\gamma(t) \tag{8.2.6}$$

Nevertheless it is possible to represent $\gamma(t)$ as a generalized stochastic process called the white noise process. The process is generalized means so that it can be constructed as a probability measure on the space s, of tempered distribution on $[0, \infty]$, and not as probability measure on the much smaller space $R^{[0,\infty]}$, like an ordinary process.

The time derivative of the Wiener process (or Brownian motion) is white noise so

$$\frac{\mathrm{d}W(t)}{\mathrm{d}t} = \gamma(t) \tag{8.2.7}$$

and Eq. (8.2.6) can be rewritten as

$$\mathrm{d}N(t) = b(t, N(t))\mathrm{d}t + \sigma(t, N(t))\mathrm{d}W(t) \tag{8.2.8}$$

This is called a stochastic differential equation of *Itô* type.

Result The one-dimensional *Itô* formula.

Let X_t be an *Itô* process given by

$$\mathrm{d}x_t = u\,\mathrm{d}t + v\,\mathrm{d}W(t) \tag{8.2.9}$$

and $g(t,x) \in C^2([0,\infty) \times R)$ (i.e., g is twice continuously differentiable on $([0, \infty) \times R)$ then

$$Y_t = g(t, X_t) \tag{8.2.10}$$

is again an *Itô* process, and

$$\mathrm{d}Y_t = \frac{\partial g}{\partial t}\mathrm{d}t + \frac{\partial g}{\partial x_t}\mathrm{d}x_t + \frac{1}{2}\frac{\partial^2 g}{\partial x_t^2}(\mathrm{d}x_t)^2 \tag{8.2.11}$$

where $(\mathrm{d}x_t)^2 = (\mathrm{d}x_t \cdot \mathrm{d}x_t)$ is computed according to the rules

$$\mathrm{d}t \cdot \mathrm{d}t = \mathrm{d}t \cdot \mathrm{d}W(t) = \mathrm{d}W(t) \cdot \mathrm{d}t = 0 \quad \text{and} \quad \mathrm{d}W(t) \cdot \mathrm{d}W(t) = \mathrm{d}t \tag{8.2.12}$$

Solution of Eq. (8.2.1) with $a(t) = b(t) + \sigma\gamma(t)$, σ is a constant representing the magnitude of the irregular fluctuations and $\gamma(t)$ is a standardized Gaussian white noise, assuming $b(t) = b$ (constant), i.e.,

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = bN(t) + \sigma\gamma(t)N(t) \tag{8.2.13}$$

or

$$\mathrm{d}N(t) = bN(t)\mathrm{d}t + \sigma N(t)\mathrm{d}W(t) \tag{8.2.14}$$

is

$$N(t) = N_0 \mathrm{e}^{(b-1/2)t+\sigma W(t)} \tag{8.2.15}$$

8.3 Stochastic Differential Equation Based Software Reliability Models

A number of NHPP based SRGM have been discussed in the previous chapters. These models treat the event of software fault detection/removal in the testing and operational phase as a counting process in discrete state space. If the size of software system is large, the number of software faults detected during the testing phase becomes large, and the change of the number of faults, which are detected and removed through debugging activities, becomes sufficiently small compared with the initial fault content at the beginning of the testing phase. Therefore,

in such a situation, the software fault detection process can be well described by a stochastic process with a continuous state space.

Under the general assumptions of NHPP software reliability growth models, i.e.,

1. Failure observation phenomenon is modeled by NHPP.
2. Failures are observed during execution caused by remaining faults in the software.
3. Each time a failure is observed, an immediate effort takes place to find the cause of the failure and the isolated faults are removed prior to future test occasions.
4. All faults in the software are mutually independent.
5. The debugging process is perfect and no new fault is introduced during debugging.

The following linear differential equation

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = b(t)[a - N(t)] \tag{8.3.1}$$

is used to describe the fault detection process, where $b(t)$ is fault detection rate per remaining fault and is a non-negative function

The testing progress is influenced by various factors all of which may not be deterministic in nature such as the testing effort expenditure, testing efficiency and skill, testing methods and strategy and so on. In order to account the uncertain factors influencing the testing process we should consider the fact that the behavior of $b(t)$ is influenced by these random factors. Hence we extend the differential Eq. (8.3.1) to the realistic equation that reflects the stochastic property of the testing process. Assuming irregular fluctuations in $b(t)$ this basic differential equation can be extended as the following stochastic differential equation

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \{b(t) + \sigma\gamma(t)\}\{a - N(t)\} \tag{8.3.2}$$

where σ is the constant representing a magnitude of irregular fluctuation and $\sigma\gamma(t)$ is a standardized Gaussian white noise. We extend the above equation to the following stochastic differential equation of an $Ito\hat{O}$ type

$$\mathrm{d}N(t) = \left\{b(t) - \frac{1}{2}\sigma^2\right\}\{a - N(t)\}\mathrm{d}t + \sigma[a - N(t)]\mathrm{d}W(t) \tag{8.3.3}$$

Using (8.2.9), (8.2.10) and (8.2.12), let

$$Y_t = g(t, N(t)) = \mathrm{Ln}(a - N(t)) \tag{8.3.4}$$

then

$$dY_t = d(\text{Ln}(a - N(t))) = -\frac{dN(t)}{(a - N(t))} - \frac{(dN(t))^2}{2(a - N(t))^2} \tag{8.3.5}$$

Using (8.2.12) we get

$$\begin{aligned}(dN(t))^2 &= \sigma^2(a - N(t))^2 dW(t) \cdot dW(t) \Rightarrow (dN(t))^2 \\ &= \sigma^2(a - N(t))^2 dt \text{ as } dW(t) \cdot dW(t) = dt\end{aligned} \tag{8.3.6}$$

Now, from (8.3.5) and using the equation of $(dN(t))^2$, we have

$$\frac{dN(t)}{a - N(t)} = -d(\text{Ln}(a - N(t))) - \frac{1}{2}\sigma^2 dt \tag{8.3.7}$$

Integrating

$$\int_0^t \frac{dN(t)}{a - N(t)} = -\int_0^t d(\text{Ln}(a - N(t))) - \int_0^t \frac{1}{2}\sigma^2 \, dt$$

$$\Rightarrow \int_0^t \frac{dN(t)}{a - N(t)} = \int_0^t b(t)dt - \frac{1}{2}\sigma^2 t + \sigma W(t)$$

$$-\text{Ln}(a - N(t))_0^t - \frac{1}{2}\sigma^2 t = \int_0^t b(t)dt - \frac{1}{2}\sigma^2 t + \sigma W(t)$$

$$-[\text{Ln}(a - N(t)) - \text{Ln}(a)] = \int_0^t b(t)dt + \sigma W(t)$$

$$\text{Ln}\left(\frac{a - N(t)}{a}\right) = -\int_0^t b(t)dt - \sigma W(t)$$

$$1 - \frac{N(t)}{a} = e^{-\int_0^t b(t)dt - \sigma W(t)}$$

$$N(t) = a\left[1 - e^{-\int_0^t b(t)dt - \sigma W(t)}\right] \tag{8.3.8}$$

Equation (8.3.8) gives the general solution of an SDE based SRGM of type (8.3.2) under the initial condition $N(0) = 0$. Several SDE based SRGM

corresponding to the various NHPP based SRGM developed ignoring the noise factor in the fault detection rate can be obtained from (8.3.8) substituting a suitable form for the non-random factor of fault detection rate $b(t)$.

8.3.1 Obtaining SRGM from the General Solution

8.3.1.1 Exponential SDE Model

An initial attempt in SDE based software reliability growth modelling was made due to Yamada et al. [3] who derived an exponential type SDE based SRGM. If we assume $b(t) = b$ in (8.3.8) then we obtain

$$N(t) = a\left(1 - \mathrm{e}^{-bt-\sigma W(t)}\right) \tag{8.3.9}$$

Taking expectation on both sides of Eq. (8.3.9) we have

$$E[N(t)] = E\left[a\left(1 - \mathrm{e}^{-bt-\sigma W(t)}\right)\right] \tag{8.3.10}$$

$$E[N(t)] = a - \left(a\,\mathrm{e}^{-bt}E\left[\mathrm{e}^{-\sigma W(t)}\right]\right) \tag{8.3.11}$$

Consider

$$\begin{aligned}
E\left[\mathrm{e}^{-\sigma W(t)}\right] &= \left[\int_{-\infty}^{\infty} \mathrm{e}^{-\sigma W(t)} \frac{1}{\sqrt{2\pi t}} \mathrm{e}^{-\left\{W(t)^2/2t\right\}}\right] \mathrm{d}W(t) \\
&= \frac{1}{\sqrt{2\pi t}}\left[\int_{-\infty}^{\infty} \mathrm{e}^{\left(W(t)^2+2\sigma t W(t)+\sigma^2 t^2-\sigma^2 t\right)/2t}\right] \mathrm{d}W(t) \\
&= \frac{1}{\sqrt{2\pi t}}\left[\int_{-\infty}^{\infty} \mathrm{e}^{\left((W(t)+\sigma t)^2\right)/2t} \mathrm{e}^{1/2(\sigma^2 t)}\right] \mathrm{d}W(t) \\
&= \mathrm{e}^{1/2(\sigma^2 t)}\left[\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi t}} \mathrm{e}^{-1/2t\left((W(t)+\sigma t)^2\right)}\left[\frac{1}{2t}\right]\right] \mathrm{d}W(t) \\
&= \mathrm{e}^{1/2(\sigma^2 t)}
\end{aligned}$$

Substituting in (8.3.11) we get

$$E[N(t)] = a\left(1 - \mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right) \tag{8.3.12}$$

Equation (8.3.12) defines the mean value function of an exponential SRGM accounting the noise factor in the fault detection rate. On similar lines, we can compute the mean value function corresponding to the various SRGM existing in the literature accounting the noise factor in the rate functions.

8.3.1.2 SDE Model for Some Other Popular NHPP Models

Delayed S-Shaped SDE Model

If we define $b(t)$ in (8.3.8) as

$$b(t) = \frac{b^2 t}{(1 + bt)}$$

then we obtain the delayed S-shaped SDE model [4]

$$N(t) = a\left[1 - (1 + bt)\mathrm{e}^{-bt - \sigma W(t)}\right] \tag{8.3.13}$$

and the expected value of $N(t)$ is given by

$$E[N(t)] = a\left(1 - (1 + bt)\mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right) \tag{8.3.14}$$

Flexible SDE Model

The flexible SDE model due to Yamada et al. [4] is given as

$$N(t) = a\left[1 - \frac{(1 + \beta)}{(1 + \beta\,\mathrm{e}^{-bt})}\mathrm{e}^{-bt - \sigma W(t)}\right] \tag{8.3.15}$$

hence the mean value function of the SRGM is

$$E[N(t)] = a\left[1 - \frac{(1 + \beta)}{(1 + \beta\,\mathrm{e}^{-bt})}\mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right] \tag{8.3.16}$$

Three-Stage SDE Model

The three-stage SDE model which describes the fault detection and correction as a three-stage process namely—fault detection, fault isolation and removal can be obtained if we define

$$b(t) = \frac{b^3 t^2}{2(1 + bt + (b^2 t^2/2))}$$

which gives

$$N(t) = a\left[1 - \left(1 + bt + \frac{b^2 t^2}{2}\right)\mathrm{e}^{-bt - \sigma W(t)}\right] \tag{8.3.17}$$

and the expected value is given as

$$E[N(t)] = a\left[1 - \left(1 + bt + \frac{b^2 t^2}{2}\right)\mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right] \tag{8.3.18}$$

8.3.2 Software Reliability Measures

8.3.2.1 Instantaneous MTBF

Let $\Delta N(t)$ be the change of $N(t)$ in the time interval $[t, t + \Delta t]$, the quantity $(\Delta t/\Delta N(t))$ gives an average fault detection interval (or average time interval between software failures) in the infinitesimal time interval $[t, t + \Delta t]$. Thus, the limit of $[t, t + \Delta t]$ such that

$$\lim_{\Delta t \to 0}\left(\frac{\Delta t}{\Delta N(t)}\right) = \frac{\mathrm{d}t}{\mathrm{d}N(t)} = \frac{1}{\mathrm{d}N(t)/\mathrm{d}t} \tag{8.3.19}$$

gives an instantaneous time interval for failure occurrences.

Then the instantaneous mean time between failures (MTBF) is given as the expected value of (8.3.19), given as

$$\mathrm{MTBF}_I(t) = E\left[\frac{1}{\mathrm{d}N(t)/\mathrm{d}t}\right] \tag{8.3.20}$$

For simplicity it is approximated as

$$\mathrm{MTBF}_I(t) = \frac{1}{E[\mathrm{d}N(t)/\mathrm{d}t]} \tag{8.3.21}$$

Now from (8.3.3)

$$\mathrm{d}N(t) = \left[b(t) - \frac{1}{2}\sigma^2\right][a - N(t)]\mathrm{d}t + \sigma[a - N(t)]\mathrm{d}W(t)$$

and

$$N(t) = a\left[1 - \mathrm{e}^{\left(-\int_0^t b(x)\mathrm{d}x - \sigma W(t)\right)}\right]$$

hence

$$E[\mathrm{d}N(t)] = \left(b(t) - \frac{1}{2}\sigma^2\right)E\left[a\,\mathrm{e}^{\left(-\int_0^t b(x)\mathrm{d}x - \sigma W(t)\right)}\mathrm{d}t\right]$$

since the Wiener process has the independent increment property, $W(t)$ and $\mathrm{d}W(t)$ are statistically independent with each other and $E[\mathrm{d}W(t)] = 0$.

Further

$$E[\mathrm{d}N(t)] = a\left(b(t) - \frac{1}{2}\sigma^2\right)\mathrm{e}^{\left(-\int_0^t b(x)\mathrm{d}x\right)}\mathrm{e}^{\left(1/2(\sigma^2 t)\right)}E[\mathrm{d}t]$$

which implies

$$\mathrm{MTBF}_I(t) = \frac{1}{a[b(t) - 1/2(\sigma^2)]\mathrm{e}^{\left(-\left[\int_0^t b(x)dx - 1/2(\sigma^2 t)\right]\right)}} \tag{8.3.22}$$

Now using (8.3.22) we can compute the instantaneous MTBF for the various SDE based models discussed above.

Exponential SDE Model

$$\mathrm{MTBF}_I(t) = \frac{1}{a[b - 1/2(\sigma^2)]\mathrm{e}^{-[b-1/2(\sigma^2)]t}} \tag{8.3.23}$$

Delayed S-Shaped SDE Model

$$\mathrm{MTBF}_I(t) = \frac{1}{a(1+bt)[(b^2t/1+bt) - 1/2(\sigma^2)]\mathrm{e}^{-[b-1/2(\sigma^2)]t}} \tag{8.3.24}$$

Flexible SDE Model

$$\mathrm{MTBF}_I(t) = \frac{1}{(a(1+\beta))/(1+\beta\,\mathrm{e}^{-bt})(b/(1+\beta\,\mathrm{e}^{-bt}) - 1/2(\sigma^2))\mathrm{e}^{-(b-1/2(\sigma^2))t}} \tag{8.3.25}$$

Three-Stage SDE Model

$$\mathrm{MTBF}_I(t) = \frac{1}{a(1+bt+(b^2t^2/2))[b^3t^2/(1+bt+b^2t^2) - 1/2(\sigma^2)]\mathrm{e}^{-[b-1/2(\sigma^2)]t}} \tag{8.3.26}$$

8.3.2.2 Cumulative MTBF

The quantity $(t/N(t))$ gives an average fault detection time interval (or average time interval between software failures) per one fault up to time t. Thus its expected value gives the MTBF measured from the initial time of testing phase up to the testing time t, called cumulative MTBF, given as,

$$\mathrm{MTBF}_c(t) = E\left[\frac{t}{N(t)}\right] \tag{8.3.27}$$

It is approximated as

$$\mathrm{MTBF}_I(t) = \frac{t}{E[N(t)]} \tag{8.3.28}$$

hence the cumulative MTBF of SDE based SRGM discussed above are obtained as follows.

Exponential SDE Model

$$\mathrm{MTBF}_c(t) = \frac{t}{a[1-\mathrm{e}^{(-(bt-1/2(\sigma^2t)))}]} \tag{8.3.29}$$

Delayed S-Shaped SDE Model

$$\mathrm{MTBF}_c(t) = \frac{t}{a[1-(1+bt)\mathrm{e}^{-(bt-1/2(\sigma^2t))}]} \tag{8.3.30}$$

Flexible SDE Model

$$\mathrm{MTBF}_c(t) = \frac{t}{a[1-(1+\beta)/(1+\beta\,\mathrm{e}^{-bt})\,\mathrm{e}^{-(bt-1/2(\sigma^2t))}]} \tag{8.3.31}$$

Three-Stage SDE Model

$$\mathrm{MTBF}_c(t) = \frac{t}{a[1-(1+bt+(b^2t^2/2))\mathrm{e}^{-(bt-1/2(\sigma^2 t))}]} \tag{8.3.32}$$

Large value of $\mathrm{MTBF_c}(t)$ depicts a high level of achieved reliability.

8.4 SDE Models Considering Fault Complexity and Distributed Development Environment

8.4.1 The Fault Complexity Model

Different fault complexity based SRGM can be formulated using the SDE models discussed in the previous sections. Kapur et al. [5] used exponential SDE model to describe the failure and removal phenomena of simple faults, delayed S-shaped SDE Model for the hard faults and three-stage SDE model to describe the complex faults. The total fault removal phenomenon of the fault complexity model is hence given as

$$\begin{aligned} N(t) = {} & a_1\left[1-\mathrm{e}^{-b_1 t-\sigma_1 W(t)}\right] + a_2\left[1-(1+b_2 t)\mathrm{e}^{-b_2 t-\sigma_2 W(t)}\right] \\ & + a_3\left[1-\left(1+b_3 t+\frac{b_3^2 t^2}{2}\right)\mathrm{e}^{-b_3 t-\sigma_3 W(t)}\right] \end{aligned} \tag{8.4.1}$$

where $a_1 = ap_1$, $a_2 = ap_2$, $a_3 = ap_3$, and $p_1 + p_2 + p_3 = 1$ and p_i is the proportion of ith type of fault in the total fault content a.

The expected value of $N(t)$ is given as

$$E[N(t)] = E[N_1(t) + N_2(t) + N_3(t)] \tag{8.4.2}$$

i.e.,

$$\begin{aligned} E[N(t)] = {} & a_1\left(1-\mathrm{e}^{-\left(b_1 t-1/2(\sigma_1^2 t)\right)}\right) + a_2\left(1-(1+b_2 t)\mathrm{e}^{-\left(b_2 t-1/2\left(\sigma_2^2 t\right)\right)}\right) \\ & + a_3\left[1-\left(1+b_3 t+\frac{b_3^2 t^2}{2}\right)\mathrm{e}^{-\left(b_3 t-1/2\left(\sigma_3^2 t\right)\right)}\right] \end{aligned} \tag{8.4.3}$$

The various reliability measures such as instantaneous MTBF and cumulative MTF for different types of faults are defined as in Sect. 8.3.2. The model is further extended [6] to incorporate the learning effect of the testing and debugging teams.

8.4.2 The Fault Complexity Model Considering Learning Effect

In Chap. 2 we have discussed the NHPP fault complexity SRGM considering the learning effect in the discrete state space. The fault removal rates for simple, hard and complex faults for that model are computed respectively as follows

$$b_1(t) = b_1, b_2(t) = b_2\left(\frac{1}{1+\beta_2 e^{-b_2 t}} - \frac{1}{1+\beta_2 + b_2 t}\right)$$

and

$$b_3(t) = b_3\left(\frac{1}{1+\beta_3 e^{-b_3 t}} - \frac{1+b_3 t}{1+\beta_3 + b_3 t + (b_3 t)^2}\right) \tag{8.4.4}$$

Using these forms of $b(t)$ in (8.3.8) we can derive the fault complexity based SDE model in the presence of learning effect. Substituting the forms of $b(t)$ given in (8.4.4) in Eq. (8.4.4) we obtain the number of faults removed for simple, hard and complex faults, given as

$$N_1(t) = a_1\left[1 - \left\{e^{-b_1 t - \sigma_1 W_1(t)}\right\}\right]$$

$$N_2(t) = a_2\left[1 - \frac{(1+\beta_2 + b_2 t)\left\{e^{-b_2 t - \sigma_2 W_2(t)}\right\}}{1+\beta_2 e^{-b_2 t}}\right]$$

and

$$N_3(t) = a_3\left[1 - \frac{(1+\beta_3 + b_3 t + b_3^2 t^2/2)\left\{e^{-b_3 t - \sigma_3 W_3(t)}\right\}}{1+\beta_3\, e^{-b_3 t}}\right] \tag{8.4.5}$$

The total fault removal phenomenon of the fault complexity model is

$$N(t) = N_1(t) + N_2(t) + N_3(t)$$

hence the mean value function of the total fault removal phenomenon is

$$E(N(t)) = a_1\left[1 - \left\{e^{\left(-b_1 t + (\sigma_1^2 t/2)\right)}\right\}\right] + a_2\left[1 - \frac{(1+\beta_2 + b_2 t)e^{\left(-b_2 t + \sigma_2^2 t/2\right)}}{1+\beta_2 e^{-b_2 t}}\right]$$
$$+ a_3\left[1 - \frac{(1+\beta_3 + b_3 t + b_3^2 t^2/2)e^{\left(-b_3 t + \sigma_3^2 t/2\right)}}{1+\beta_3\, e^{-b_3 t}}\right] \tag{8.4.6}$$

Now using the results of Sect. 8.3.2 we obtain the instantaneous MTBF and cumulative MTBF for the different types of faults.

8.4.2.1 Simple Faults

$$\mathrm{MTBF}_I(t) = \frac{1}{a_1(b_1 - 1/2(\sigma^2))\mathrm{e}^{-(b_1 - 1/2(\sigma^2))t}} \tag{8.4.7}$$

$$\mathrm{MTBF}_C(t) = \frac{t}{a_1\left[1 - \mathrm{e}^{-(b_1 - 1/2(\sigma^2))t}\right]} \tag{8.4.8}$$

8.4.2.2 Hard Faults

$$\mathrm{MTBF}_I(t) = \frac{1}{a_2\left[\frac{1+\beta+b_2 t}{1+\beta\mathrm{e}^{-b_2 t}}\right]\left[\frac{\left(b_2(1+\beta+b_2 t) - b_2\left(1+\beta\mathrm{e}^{-\mathrm{b}_2\mathrm{t}}\right)\right)}{((1+\beta+b_2 t)(1+\beta\mathrm{e}^{-b_2 t}))} - \frac{1}{2}\sigma_2^2\right]\mathrm{e}^{-\left(b_2 - \frac{1}{2}\sigma_2^2\right)t}} \tag{8.4.9}$$

$$\mathrm{MTBF}_C(t) = \frac{t}{a_2\left[1 - \frac{(1+\beta+b_2 t)\mathrm{e}^{-\left(\mathrm{b}_2\mathrm{t} - \sigma_2^2\mathrm{t}/2\right)}}{1+\beta\,\mathrm{e}^{-b_2 t}}\right]} \tag{8.4.10}$$

8.4.2.3 Complex Faults

$$\mathrm{MTBF}_I(t) = \frac{1}{a_3\left[\frac{S}{1+\beta\mathrm{e}^{-b_3 t}}\right]\left[\frac{(b_3(S) - b_3(1+\beta\mathrm{e}^{-\mathrm{b}_3\mathrm{t}})(1+b_3 t))}{((S)(1+\beta\mathrm{e}^{-b_3 t}))} - \frac{\sigma_3^2}{2}\right]\mathrm{e}^{-\left(b_3 - \frac{\sigma_3^2}{2}\right)t}} \tag{8.4.11}$$

where

$$S = 1 + \beta + b_3 t + \frac{b_3^2 t^2}{2}$$

$$\mathrm{MTBF}_C(t) = \frac{t}{a_3\left[1 - \left(\left(1 + \beta + b_3 t + b_3^2 t^2/2\right)\mathrm{e}^{-\left(b_3 t - \sigma_3^2 t/2\right)}\right)\Big/\left(1 + \beta\,\mathrm{e}^{-b_3 t}\right)\right]} \tag{8.4.12}$$

8.4.3 An SDE Based SRGM for Distributed Development Environment

The fault complexity models are usually extended to describe the growth of software systems developed under the distributed development environment

(DDE). Let us assume that, the software is consisting of n used and m newly developed components. Used components are the software modules, which have been developed for some other application and have some supporting function, which is also required for the new software under consideration. The old code is used either as such or with certain modifications to suit the current need. These modules are generally assumed to contain only simple types of faults as they have been tested in their previous applications. On the other hand the newly developed components are developed anew for specific functions of the new software as such they are assumed to contain mostly hard or complex types of faults which depends on their complexity and size characteristics.

The fault detection and removal phenomenon of the used components can be described by either of the above two models developed for the simple faults and the concerned phenomenon for the new components can be described by the models corresponding to the hard and complex faults [7]. If we assume that p components of m new contain hard faults and remaining q components contain complex faults, then the mean value function of the DDE software is

$$\begin{aligned} E[N(t)] = & \sum_{i=0}^{n} a_i \left(1 - \mathrm{e}^{-\left(b_i t - 1/2\left(\sigma_i^2 t\right)\right)}\right) \\ & + \sum_{i=n+1}^{n+1+p} a_i \left(1 - (1 + b_i t)\mathrm{e}^{-\left(b_i t - 1/2\left(\sigma_i^2 t\right)\right)}\right) \\ & + \sum_{i=n+p+2}^{n+m} a_i \left[1 - \left(1 + b_i t + \frac{b_i^2 t^2}{2}\right)\mathrm{e}^{-\left(b_i t - 1/2\left(\sigma_i{}^2 t\right)\right)}\right] \end{aligned} \tag{8.4.13}$$

Incorporating the learning effect the DDE model becomes

$$\begin{aligned} E[N(t)] = & \sum_{i=0}^{n} a_i \left(1 - \mathrm{e}^{-\left(b_i t - 1/2(\sigma_i^2 t)\right)}\right) \\ & + \sum_{i=n+1}^{n+1+p} a_i \left[1 - \frac{(1 + \beta_i + b_i t)\mathrm{e}^{\left(-b_i t + \sigma_i^2 t/2\right)}}{1 + \beta_i \mathrm{e}^{-b_i t}}\right] \\ & + \sum_{i=n+p+2}^{n+m} a_i \left[1 - \frac{\left(1 + \beta_i + b_i t + b_i^2 t^2/2\right)\mathrm{e}^{\left(-b_i t + \sigma_i^2 t/2\right)}}{1 + \beta_i \mathrm{e}^{-b_i t}}\right] \end{aligned} \tag{8.4.14}$$

8.5 Change Point SDE Model

Owing to the improved estimation power of change point model, SDE based software reliability modelling is also extended to the change point models. Here we show the development of change point models for exponential, delayed S-shaped and flexible SDE Models [8].

8.5.1 Exponential Change Point SDE Model

The fault detection rates with random factor for the change point models are

$$b(t) = \begin{cases} b_1 + \sigma\gamma(t) & 0 \le t \le \tau \\ b_2 + \sigma\gamma(t) & t > \tau \end{cases} \tag{8.5.1}$$

The stochastic differential equation for the model is then formulated as

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \begin{cases} (b_1 + \sigma\gamma(t))[a - N(t)] & 0 \le t \le \tau \\ (b_2 + \sigma\gamma(t))[a - N(t)] & t > \tau \end{cases} \tag{8.5.2}$$

The transition probability distribution of this model is

$$N(t) = \begin{cases} a\left[1 - \mathrm{e}^{(-b_1 t - \sigma W(t))}\right] & 0 \le t \le \tau \\ a\left[1 - \mathrm{e}^{(-b_1\tau - b_2(t-\tau) - \sigma W(t))}\right] & t > \tau \end{cases} \tag{8.5.3}$$

The mean number of detected faults is obtained taking expectation of $N(t)$

$$E[N(t)] = \begin{cases} a\left[1 - \mathrm{e}^{\left(-b_1 t + (\sigma^2 t)/2\right)}\right] & 0 \le t \le \tau \\ a\left[1 - \mathrm{e}^{\left(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2\right)}\right] & t > \tau \end{cases} \tag{8.5.4}$$

Following the results of Sect. 8.3.2 the instantaneous and cumulative MTBF are obtained

$$\mathrm{MTBF}_I(t) = \begin{cases} \dfrac{1}{a[b_1 - 1/2(\sigma^2 \mathrm{e}^{(-b_1 t + (\sigma^2 t)/2)})]} & 0 \le t \le \tau \\ \dfrac{1}{a[b_2 - 1/2(\sigma^2 \mathrm{e}^{(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2)})]} & t > \tau \end{cases} \tag{8.5.5}$$

$$\mathrm{MTBF}_C(t) = \begin{cases} \dfrac{t}{a[1 - \mathrm{e}^{(-b_1 t + (\sigma^2 t)/2)}]} & 0 \le t \le \tau \\ \dfrac{t}{a[1 - \mathrm{e}^{(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2)}]} & t > \tau \end{cases} \tag{8.5.6}$$

8.5.2 Delayed S-Shaped Change Point SDE Model

The following S-shaped random fault detection rates describe the failure and removal process by a delayed S-shaped curve

$$b(t) = \begin{cases} \dfrac{b_1^2 t}{1 + b_1 t} + \sigma\gamma(t) & 0 \le t \le \tau \\ \dfrac{b_2^2 t}{1 + b_2 t} + \sigma\gamma(t) & t > \tau \end{cases} \tag{8.5.7}$$

Accordingly the stochastic SDE model is formulated as

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \begin{cases} \left[\dfrac{b_1^2 t}{1+b_1 t} + \sigma\gamma(t)\right][a - N(t)] & 0 \le t \le \tau \\ \left[\dfrac{b_2^2 t}{1+b_2 t} + \sigma\gamma(t)\right][a - N(t)] & t > \tau \end{cases} \tag{8.5.8}$$

Therefore, the transition probability distribution of this model is obtained as follows

$$N(t) = \begin{cases} a\left[1 - (1+b_1 t)\mathrm{e}^{(-b_1 t - \sigma W(t))}\right] & 0 \le t \le \tau \\ a\left[1 - \dfrac{(1+b_1\tau)}{(1+b_2\tau)}(1+b_2 t)\mathrm{e}^{(-b_1\tau - b_2(t-\tau) - \sigma W(t))}\right] & t > \tau \end{cases} \tag{8.5.9}$$

The mean number of detected faults is given by

$$E[N(t)] = \begin{cases} a\left[1 - (1+b_1 t)\exp^{\left(-b_1 t + (\sigma^2 t)/2\right)}\right] & 0 \le t \le \tau \\ a\left[1 - \left(\dfrac{(1+b_1\tau)}{(1+b_2\tau)}\right)\begin{pmatrix} 1 \\ +b_2 t \end{pmatrix}\mathrm{e}^{\left(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2\right)}\right] & t > \tau \end{cases} \tag{8.5.10}$$

and Eqs. (8.5.11) and (8.5.12) give the instantaneous and cumulative values of the MTBF

$$\mathrm{MTBF}_I(t) = \begin{cases} \dfrac{1}{a\left[(1+b_1 t)\left(\frac{b_1^2 t}{1+b_1 t} - \frac{1}{2}\sigma^2\right)\mathrm{e}^{\left(-b_1 t + \frac{\sigma^2 t}{2}\right)}\right]} & 0 \le t \le \tau \\ \dfrac{1}{a\left[\left(\frac{(1+b_1\tau)}{(1+b_2\tau)}\right)\begin{pmatrix} 1 \\ +b_2 t \end{pmatrix}\begin{pmatrix} \frac{b_2^2 t}{1+b_2 t} \\ -1/2\sigma^2 \end{pmatrix}\mathrm{e}^{\begin{pmatrix} -b_1\tau + \frac{\sigma^2 t}{2} \\ -b_2(t-\tau) \end{pmatrix}}\right]} & t > \tau \end{cases} \tag{8.5.11}$$

$$\mathrm{MTBF}_C(t) = \begin{cases} \dfrac{t}{a[1 - (1+b_1 t)\mathrm{e}^{(-b_1 t + (\sigma^2 t)/2)}]} & 0 \le t \le \tau \\ \dfrac{t}{a[1 - ((1+b_1\tau)/(1+b_2\tau))(1+b_2 t)\mathrm{e}^{(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2)}]} & t > \tau \end{cases} \tag{8.5.12}$$

8.5.3 Flexible Change Point SDE Model

Fault detection rates for the flexible SDE Model for reliability growth measurement incorporating the effect of random factors are defined by Eq. (8.5.13)

$$b(t) = \begin{cases} \dfrac{b_1}{1+\beta e^{-b_1 t}} + \sigma\gamma(t) & 0 \le t \le \tau \\ \dfrac{b_2}{1+\beta e^{-b_2 t}} + \sigma\gamma(t) & t > \tau \end{cases} \tag{8.5.13}$$

Here β is assumed to be same before and after the change point for the sake of simplicity. The intensity function of the stochastic model is hence formulated according to the following stochastic differential equation

$$\frac{dN(t)}{dt} = \begin{cases} \left[\dfrac{b_1}{1+\beta e^{-b_1 t}} + \sigma\gamma(t)\right][a - N(t)] & 0 \le t \le \tau \\ \left[\dfrac{b_2}{1+\beta e^{-b_2 t}} + \sigma\gamma(t)\right][a - N(t)] & t > \tau \end{cases} \tag{8.5.14}$$

The transition probability distribution of this model is obtained as follows

$$N(t) = \begin{cases} a\left[1 - \dfrac{(1+\beta)}{1+\beta\, e^{-b_1 t}} e^{(-b_1 t - \sigma W(t))}\right] & 0 \le t \le \tau \\ a\left[1 - \left(\dfrac{(1+\beta)\left(1+\beta\, e^{-b_2\tau}\right)/\left(1+\beta\, e^{-b_1\tau}\right)\left(1+\beta\, e^{-b_2 t}\right)}{e^{(-b_1\tau - b_2(t-\tau) - \sigma W(t))}}\right)\right] & t > \tau \end{cases} \tag{8.5.15}$$

The expected value of the removal process is given as

$$E[N(t)] = \begin{cases} a\left[1 - \dfrac{(1+\beta)}{(1+\beta\, e^{-b_1 t})} e^{\left(-b_1 t + (\sigma^2 t)/2\right)}\right] & 0 \le t \le \tau \\ a\left[1 - \dfrac{(1+\beta)\left(1+\beta e^{-b_2\tau}\right)}{(1+\beta e^{-b_1\tau})(1+\beta e^{-b_2 t})} e^{\begin{pmatrix} -b_1\tau + (\sigma^2 t)/2 \\ -b_2(t-\tau) \end{pmatrix}}\right] & t > \tau \end{cases} \tag{8.5.16}$$

The MTBF functions for this model are

$$\mathrm{MTBF}_I(t) \begin{cases} \dfrac{1}{a\left[\frac{(1+\beta)}{(1+\beta e^{-b_1 t})}\left(\frac{b_1}{1+\beta e^{-b_1 t}} - \frac{1}{2}\sigma^2\right) e^{\left(-b_1 t + \frac{\sigma^2 t}{2}\right)}\right]} & 0 \le t \le \tau \\ \dfrac{t}{a\left[\frac{(1+\beta)\left(1+\beta e^{-b_2\tau}\right)}{\left(1+\beta e^{-b_1\tau}\right)\left(1+\beta e^{-b_2 t}\right)}\left(\frac{\left(\frac{b_2}{1+\beta e^{-b_2 t}} - \frac{1}{2}\sigma^2\right)}{e^{\left(-b_1\tau - b_2(t-\tau) + \frac{\sigma^2 t}{2}\right)}}\right)\right]} & t > \tau \end{cases} \tag{8.5.17}$$

$$\mathrm{MTBF}_C(t) = \begin{cases} \dfrac{t}{a[1-(1+\beta)/(1+\beta e^{-b_1 t}) e^{(-b_1 t + (\sigma^2 t)/2)}]} & 0 \le t \le \tau \\ \dfrac{t}{a[1-(1+\beta)(1+\beta e^{-b_2\tau})/(1+\beta e^{-b_1\tau})(1+\beta e^{-b_2 t}) e^{(-b_1\tau - b_2(t-\tau) + (\sigma^2 t)/2)}]} & t > \tau \end{cases} \tag{8.5.18}$$

8.6 SDE Based Testing Domain Models

In Chap. 4, we have developed a number of functions, which can describe the growth of testing domain, and then used the domain functions to analyze the software reliability. The software reliability growth models, which measure the measure of reliability with respect to the testing domain growth, can be used to obtain the domain growth dependent fault detection functions, which can then be used in (8.3.8) to obtain the SRGM with random factor.

8.6.1 SRGM Development: Basic Testing Domain

Refer to Sect. 4.3.1, expected number of faults detected is expressed as

$$m_b(t) = a\left(1 + \frac{b\,\mathrm{e}^{-vt} - v\,\mathrm{e}^{-bt}}{v - b}\right); \quad v \neq b \tag{8.6.1}$$

in the basic testing domain dependent exponential SRGM.

The above SRGM describes a two-stage process, namely—testing domain isolation and fault detection. The mean value function of the SRGM model is derived formulating the intensity function as

$$\frac{\mathrm{d}}{\mathrm{d}t} m_b(t) = b(u_b(t) - m_b(t)) \tag{8.6.2}$$

whereas the basic testing domain $u_b(t)$ is obtained from the differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t} u_b(t) = v(a - u_b(t)) \tag{8.6.3}$$

The SRGM (8.6.1) can be obtained in one stage using the fault detection rate per remaining fault, obtained from

$$b_b(t) = \frac{m_b'(t)}{a - m_b(t)} \tag{8.6.4}$$

This implies

$$b_b(t) = \frac{vb\left(\mathrm{e}^{-bt} - \mathrm{e}^{-vt}\right)}{\left(v\,\mathrm{e}^{-bt} - b\,\mathrm{e}^{-vt}\right)} \tag{8.6.5}$$

and substituting in equation

$$\frac{\mathrm{d}}{\mathrm{d}t}m_b(t) = b_b(t)(u_b(t) - m_b(t)) \tag{8.6.6}$$

If we substitute $b_b(t)$ in place of $b(t)$ in (8.3.8), we obtain the basic testing domain dependent SDE model, assuming $b_b(t)$ has irregular fluctuation. Substituting and solving (8.3.8) we obtain the transition probability distribution of the exponential basic testing domain dependent SDE model [9]

$$N_b(t) = a\left[1 - \frac{\left(v\,\mathrm{e}^{-bt} - b\,\mathrm{e}^{-vt}\right)\mathrm{e}^{-\sigma W(t)}}{(v-b)}\right] \tag{8.6.7}$$

Thus the mean number of detected faults up to testing time t for the basic testing domain model is

$$E(N_b(t)) = a\left[1 - \frac{\left(v\,\mathrm{e}^{\left(-bt+\left(\sigma^2 t/2\right)\right)} - b\,\mathrm{e}^{\left(-vt+\left(\sigma^2 t/2\right)\right)}\right)}{(v-b)}\right] \tag{8.6.8}$$

8.6.2 SRGM for Testing Domain with Skill Factor

The mean value function of SRGM derived from testing domain with skill factor ignoring the random fluctuation is (refer to Sect. 4.3.1)

$$m_{s,p}(t) = a\left[1 + \frac{bp}{(v-b)}\left(vt + \frac{2v-b}{v-b}\right)\mathrm{e}^{-vt} - \left(1 + bp\frac{2v-b}{(v-b)^2}\right)\mathrm{e}^{-bt}\right] \tag{8.6.9}$$

The testing domain with skill is formulated as a two-stage process. The first stage describes the faults existing in the isolated testing domain and the second stage describes the detectable faults in the domain. With the detectable fault domain function, the SRGM can be obtained for the fault detection process. The SRGM (8.6.9) can also be obtained in one stage if we define

$$\begin{aligned} b_{s,p}(t) &= \frac{m_b'(t)}{a - m_b(t)} \\ &= \frac{b\left\{\left[(v-b)^2 + bp(2v-b)\right]\mathrm{e}^{-bt} - (1 + (v-b)t)v^2 p\,\mathrm{e}^{-vt}\right\}}{\left\{\left[(v-b)^2 + bp(2v-b)\right]\mathrm{e}^{-bt} - bp(vt(v-b) + 2v - b)\mathrm{e}^{-vt}\right\}} \end{aligned} \tag{8.6.10}$$

Using the fault detection rate $b_{s,p}(t)$ in Eq. (8.3.8) we derive the transition probability distribution $N_{s,p}(t)$ of the testing domain with skill factor dependent SDE model

$$N_{s,p}(t) = a\left[1 + \frac{bp}{(v-b)}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt}e^{-\sigma^{-\sigma W(t)}} - \left(1 + bp\frac{2v-b}{(v-b)^2}\right)e^{-bt}e^{-\sigma^{-\sigma W(t)}}\right] \tag{8.6.11}$$

The mean number of detected faults up to testing time t for the testing domain with skill factor is then given as

$$E\left(N_{s,p}(t)\right) = a\left[1 + \frac{bp}{(v-b)}\left(vt + \frac{2v-b}{v-b}\right)e^{\left(-vt+(\sigma^2 t)/2\right)} - \left(1 + bp\frac{2v-b}{(v-b)^2}\right)e^{\left(-bt+(\sigma^2 t)/2\right)}\right] \tag{8.6.12}$$

For this SRGM if we assume that the size of initial testing domain is a, i.e., no part of the testing domain can be isolated at the starting time of the testing phase, then from the testing domain function (4.3.7) and SRGM (4.3.16) we obtain the modified fault detection rate $b_{s,p}(t)$, denoted as $b_s(t)$

$$b_s(t) = \frac{bv^2\left[e^{-bt} - (1 + (v-b)t)e^{-vt}\right]}{\left[v^2e^{-bt} - b(vt(v-b) + 2v - b)e^{-vt}\right]} \tag{8.6.13}$$

so the transition probability distribution and the mean value function of the SDE model for testing domain with skill factor become

$$N_s(t) = a\left[1 - \left(\frac{v}{v-b}\right)^2 e^{-bt}e^{-\sigma W(t)} + \frac{b}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{-vt}e^{-\sigma W(t)}\right] \tag{8.6.14}$$

$$E(N_s(t)) = a\left[\begin{array}{l} 1 - \left(\frac{v}{v-b}\right)^2 e^{\left(-bt+(\sigma^2 t)/2\right)} \\ + \frac{b}{v-b}\left(vt + \frac{2v-b}{v-b}\right)e^{\left(-vt+(\sigma^2 t)/2\right)} \end{array}\right] \tag{8.6.15}$$

8.6.3 Imperfect Testing Domain Dependent SDE Based SRGM

Based on the similar analysis we obtain the single stage, fault detection rate for imperfect testing domain based SRGM derived from ordinary differential equation (Sect. 4.3.1)

$$b_i(t) = \frac{vb\left[\alpha(v-b)e^{\alpha t} - v(\alpha+b)e^{-vt} + b(\alpha+v)e^{-bt}\right]}{\left[(\alpha+v)(v-b)(\alpha+b) - vb((v-b)e^{\alpha t} + (\alpha+b)e^{-vt} - (\alpha+v)e^{-bt})\right]} \tag{8.6.16}$$

Assuming irregular fluctuations in the fault detection rate (8.6.16) yields the transition probability distribution $N_i(t)$ and SDE based SRGM $E[N_i(t)]$

$$N_i(t) = avb\left[\frac{\mathrm{e}^{-(-\alpha t+\sigma W(t))}}{(\alpha+v)(\alpha+b)} + \frac{\mathrm{e}^{-(vt+\sigma W(t))}}{(\alpha+v)(v-b)} - \frac{\mathrm{e}^{-(-bt+\sigma W(t))}}{(v-b)(\alpha+b)}\right] \tag{8.6.17}$$

$$E(N_i(t)) = avb\left[\frac{\mathrm{e}(\alpha t + (\sigma^2 t)/2)}{(\alpha+v)(\alpha+b)} + \frac{\mathrm{e}(-vt + (\sigma^2 t)/2)}{(\alpha+v)(v-b)} - \frac{e(-bt + (\sigma^2 t)/2)}{(v-b)(\alpha+b)}\right] \tag{8.6.18}$$

8.6.4 Software Reliability Measures

Using the results of the previous sections here, we define the instantaneous and cumulative MTBF for the testing domain dependent SRGM, discussed above.

8.6.4.1 Instantaneous MTBF for Basic Testing Domain Dependent SRGM

$$\mathrm{MTBF}_b^I(t) = \frac{1}{a\left[\frac{vb(\mathrm{e}^{-bt}-\mathrm{e}^{-vt})}{(v\mathrm{e}^{-bt}-b\mathrm{e}^{-vt})} - \frac{\sigma^2}{2}\right]\frac{1}{(v-b)}\left[v\mathrm{e}^{\left(-bt+\frac{\sigma^2}{2}t\right)} - b^{e\left(-vt+\frac{\sigma^2}{2}t\right)}\right]} \tag{8.6.19}$$

8.6.4.2 Instantaneous MTBF for Testing Domain with Skill Factor Dependent SRGM

$$\mathrm{MTBF}_{s,p}^I(t) = \left(a\left[\frac{b\left\{\left[(v-b)^2+bp(2v-b)\right]\mathrm{e}^{-bt} - (1+(v-b)t)v^2p\,\mathrm{e}^{-vt}\right\}}{\left\{\left[(v-b)^2+bp(2v-b)\right]\mathrm{e}^{-bt} - bp(vt(v-b)+2v-b)\mathrm{e}^{-vt}\right\}} - \frac{\sigma^2}{2}\right]\left[\begin{array}{l}\left(1+bp\frac{2v-b}{(v-b)^2}\right)\mathrm{e}^{\left(-bt+(\sigma^2/2)t\right)} \\ -\frac{bp}{(v-b)}\left(vt+\frac{2v-b}{v-b}\right)\mathrm{e}^{\left(-vt+(\sigma^2 t)/2\right)}\end{array}\right]\right)^{-1} \tag{8.6.20}$$

$$\mathrm{MTBF}_s^I(t) = \left(a\left[\frac{v^2b(\mathrm{e}^{-bt}-(1+(v-b)t)\mathrm{e}^{-vt})}{(v^2\mathrm{e}^{-bt}-b(vt(v-b)+2v-b)\mathrm{e}^{-vt})} - \frac{\sigma^2}{2}\right]\left[\left(\frac{v}{v-b}\right)^2\mathrm{e}^{\left(-bt+(\sigma^2/2)t\right)} - \frac{b}{v-b}\left(vt+\frac{2v-b}{v-b}\right)\mathrm{e}^{\left(-vt+(\sigma^2/2)t\right)}\right]\right)^{-1} \tag{8.6.21}$$

8.6.4.3 Instantaneous MTBF for Imperfect Testing Domain Dependent SRGM

$$\mathrm{MTBF}_i^I(t) = \left(\begin{array}{l} a\left[\dfrac{vb\left[\alpha(v-b)\mathrm{e}^{\alpha t} - v(\alpha+b)\mathrm{e}^{-vt} + b(\alpha+v)\mathrm{e}^{-bt}\right]}{(\alpha+v)(\alpha+b)(v-b) - vb[(v-b)\mathrm{e}^{\alpha t} + (\alpha+b)\mathrm{e}^{-vt} - (\alpha+v)\mathrm{e}^{-bt}]} - \dfrac{\sigma^2}{2}\right] \\ \left[1 - vb\left(\dfrac{\mathrm{e}^{(\alpha+(\sigma^2/2))t}}{(\alpha+v)(\alpha+b)} + \dfrac{\mathrm{e}^{-(v-(\sigma^2/2))t}}{(\alpha+v)(v-b)} - \dfrac{\mathrm{e}^{-(b-(\sigma^2/2))t}}{(v-b)(\alpha+b)}\right)\right] \end{array} \right)^{-1} \tag{8.6.22}$$

8.6.4.4 Cumulative MTBF for Basic Testing Domain Dependent SRGM

$$\mathrm{MTBF}_b^c(t) = t\left(a\left[1 - \frac{\left(v\,\mathrm{e}^{\left(-bt+\left(\sigma^2 t/2\right)\right)} - b\,\mathrm{e}^{\left(-vt+\left(\sigma^2 t/2\right)\right)}\right)}{(v-b)}\right]\right)^{-1} \tag{8.6.23}$$

8.6.4.5 Cumulative MTBF for Testing Domain with Skill Factor Dependent SRGM

$$\mathrm{MTBF}_{s,p}^c(t) = t\left(a\left[\begin{array}{l} 1 + \dfrac{bp}{(v-b)}\left(vt + \dfrac{2v-b}{v-b}\right)\mathrm{e}^{\left(-vt+(\sigma^2 t/2)\right)} \\ \quad -\left(1 + bp\dfrac{2v-b}{(v-b)^2}\right)\mathrm{e}^{\left(-bt+(\sigma^2 t/2)\right)} \end{array}\right]\right)^{-1} \tag{8.6.24}$$

$$\mathrm{MTBF}_s^c(t) = t\left(a\left[1 - \left(\frac{v}{v-b}\right)^2 \mathrm{e}^{\left(-bt+(\sigma^2 t/2)\right)} + \frac{b}{v-b}\left(vt + \frac{2v-b}{v-b}\right)\mathrm{e}^{\left(-vt+(\sigma^2 t/2)\right)}\right]\right)^{-1} \tag{8.6.25}$$

8.6.4.6 Cumulative MTBF for Imperfect Testing Domain Dependent SRGM

$$\mathrm{MTBF}_i^c(t) = t\left(avb\left[\frac{\mathrm{e}^{\left(\alpha t+(\sigma^2 t/2)\right)}}{(\alpha+v)(\alpha+b)} + \frac{\mathrm{e}^{\left(-vt+(\sigma^2 t/2)\right)}}{(\alpha+v)(v-b)} - \frac{\mathrm{e}^{\left(-bt+(\sigma^2 t/2)\right)}}{(v-b)(\alpha+b)}\right]\right)^{-1} \tag{8.6.26}$$

8.7 Data Analysis and Parameter Estimation

In this chapter we have discussed various stochastic differential equation based SRGM describing different aspects of testing process viz. uniform and non-uniform operational profile, S-shaped, flexible, fault complexity based, change point and testing domain dependent models. Now we show an application of these models by estimating the parameters of the various models discussed in the chapter and predicting the testing process by means of the goodness of fit curves.

Failure Data Set

The failure data is obtained during testing of software that runs on an element within a wireless network switching centre. Its main functions include routing voice channels, signaling messages to relevant radio resources and processing entities within the switching centre. Multiple systems were used in parallel to test the software. The software reliability data was obtained [10] by aggregating (on a weekly basis) the test time and the number of failures across all the test systems. During 34 weeks the software is tested for 1,001 days and a total of 181 failures were observed.

Following SDE models have been chosen for data analysis and parameter estimation.

Model 1 (M1) Exponential SDE Model [3]

$$E[N(t)] = a\left(1 - \mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right)$$

Model 2 (M2) Delayed S-shaped SDE Model [4]

$$E[N(t)] = a\left(1 - (1+bt)\mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right)$$

Model 3 (M3) Flexible SDE Model [4]

$$E[N(t)] = a\left[1 - \frac{(1+\beta)}{(1+\beta\,\mathrm{e}^{-bt})}\mathrm{e}^{-\left(bt - \frac{1}{2}\sigma^2 t\right)}\right]$$

Model 4 (M4) Three-Stage SDE Model [4]

$$E[N(t)] = a\left[1 - \left(1 + bt + \frac{b^2 t^2}{2}\right)\mathrm{e}^{-\left(bt - 1/2(\sigma^2 t)\right)}\right]$$

Model 5 (M5) The Fault Complexity Model [5]

$$E[N(t)] = a_1\left(1 - e^{-\left(b_1 t - 1/2\left(\sigma_1^2 t\right)\right)}\right) + a_2\left(1 - (1 + b_2 t)e^{-\left(b_2 t - 1/2\left(\sigma_2^2 t\right)\right)}\right)$$
$$+ a_3\left[1 - \left(1 + b_3 t + \frac{b_3^2 t^2}{2}\right)e^{-\left(b_3 t - 1/2(\sigma_3^2 t)\right)}\right]$$

Model 6 (M6) The Fault Complexity Model with Learning Effect [6]

$$E(N(t)) = a_1\left[1 - \left\{e^{\left(-b_1 t + (\sigma_1^2 t/2)\right)}\right\}\right] + a_2\left[1 - \frac{(1 + \beta_2 + b_2 t)e^{\left(-b_2 t + \sigma_2^2 t/2\right)}}{1 + \beta_2 e^{-b_2 t}}\right]$$
$$+ a_3\left[1 - \frac{(1 + \beta_3 + b_3 t + b_3^2 t^2/2)e^{\left(-b_3 t + \sigma_3^2 t/2\right)}}{1 + \beta_3 e^{-b_3 t}}\right]$$

Model 7 (M7) Exponential Change Point SDE Model [8]

$$E[N(t)] = \begin{cases} a\left[1 - e^{\left(-b_1 t + (\sigma^2 t/2)\right)}\right] & 0 \le t \le \tau \\ a\left[1 - e^{\left(-b_1 \tau - b_2(t-\tau) + (\sigma^2 t/2)\right)}\right] & t > \tau \end{cases}$$

Model 8 (M8) Delayed S-shaped Change Point SDE Model [8]

$$E[N(t)] = \begin{cases} a\left[1 - (1 + b_1 t)\exp^{\left(-b_1 t + (\sigma^2 t/2)\right)}\right] & 0 \le t \le \tau \\ a\left[1 - \left(\frac{(1 + b_1 \tau)}{(1 + b_2 \tau)}\right)\begin{pmatrix} 1 \\ +b_2 t \end{pmatrix} e^{\left(-b_1 \tau - b_2(t-\tau) + (\sigma^2 t/2)\right)}\right] & t > \tau \end{cases}$$

Model 9 (M9) Basic Testing Domain based SDE SRGM [9]

$$E(N_b(t)) = a\left[1 - \frac{\left(v\, e^{\left(-bt + \left(\sigma^2 t/2\right)\right)} - b\, e^{\left(-vt + \left(\sigma^2 t/2\right)\right)}\right)}{(v - b)}\right]$$

Model 10 (M10) SDE based SRGM for Testing Domain with Skill Factor [9]

$$E\left(N_{s,p}(t)\right) = a\left[1 + \frac{bp}{(v - b)}\left(vt + \frac{2v - b}{v - b}\right)e^{\left(-vt + (\sigma^2 t/2)\right)} - \left(1 + bp\frac{2v - b}{(v - b)^2}\right)e^{\left(-bt + (\sigma^2 t/2)\right)}\right]$$

Model 11 (M11) Imperfect Testing Domain Dependent SDE based SRGM [9]

$$E(N_i(t)) = avb\left[\frac{e^{\left(\alpha t + (\sigma^2 t/2)\right)}}{(\alpha + v)(\alpha + b)} + \frac{e^{\left(-vt + (\sigma^2 t/2)\right)}}{(\alpha + v)(v - b)} - \frac{e^{\left(-bt + (\sigma^2 t/2)\right)}}{(v - b)(\alpha + b)}\right]$$

Results of parameter estimations are shown in Table 8.1. Analysis of results depicts that the exponential SDE models estimate a very high value of initial fault content in contrast to all other models with a reasonably good value of comparison

Table 8.1 Estimation results of models M1–M11

Model	Estimated parameters									Comparison criteria	
	a, a_1	a_2	a_3	b, b_1	b_2, v	b_3	σ	β, β_2, p	β_3, α	MSE	R^2
M1	1249	–	–	0.00545	–	–	0.03714	–	–	20.70	0.994
M2	225	–	–	0.08812	–	–	0.00010	–	–	21.42	0.994
M3	229	–	–	0.08761	–	–	8.95e−5	3.69	–	6.60	0.999
M4	189	–	–	0.16593	–	–	4.80e−8	–	–	68.80	0.98
M5	102	86	64	0.09598	0.08492	0.1327	0.20456	–	–	7.59	0.998
M6	55	113	40	0.09866	0.14300	0.2451	0.09812	19.54	0.0095	7.64	0.998
M7	265	–	–	0.05225	0.07265	–	0.24061	–	–	12.67	0.995
M8	234	–	–	0.08616	0.08196	–	1.04e−5	–	–	21.90	0.994
M9	350	–	–	0.02409	0.31482	–	0.00199	–	–	12.19	0.996
M10	245	–	–	0.06837	0.13863	–	0.01581	0.699	–	7.09	0.998
M11	123	–	–	0.00012	0.00015	–	0.32709	–	0.0035	292.15	0.894

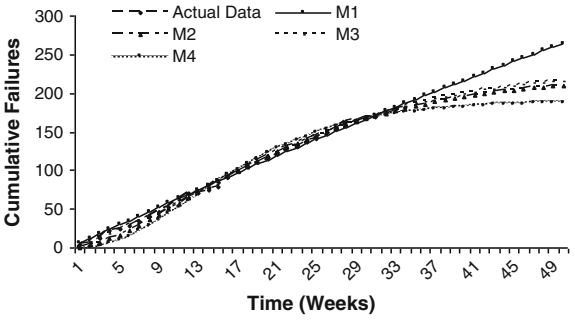


Fig. 8.1 Goodness of fit curve models M1–M4

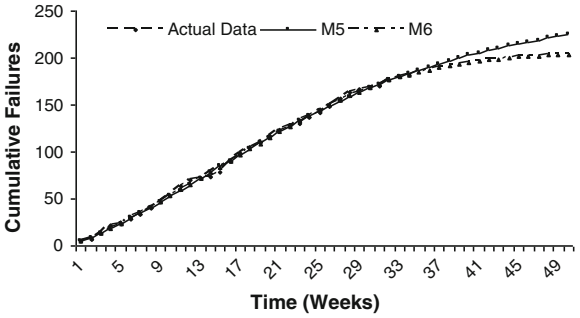


Fig. 8.2 Goodness of fit curve for Fault complexity based SDE models (M5 and M6)

criteria MSE and R^2. Flexible SDE model, fault complexity models and testing domain with skill factor based SRGM provide good fit on the data, while the flexible SDE models fits best on this data. The fitting of imperfect testing domain dependent SDE model suggests that the fault generation model cannot be applied

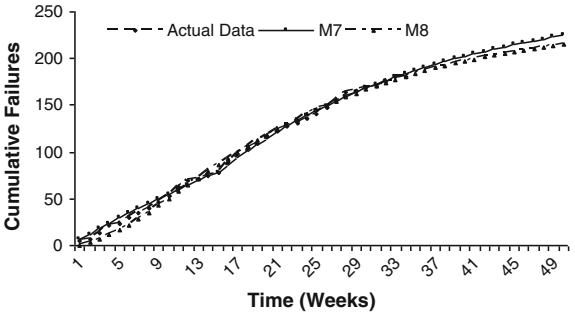


Fig. 8.3 Goodness of fit curve for change point based SDE models (M7 and M8)

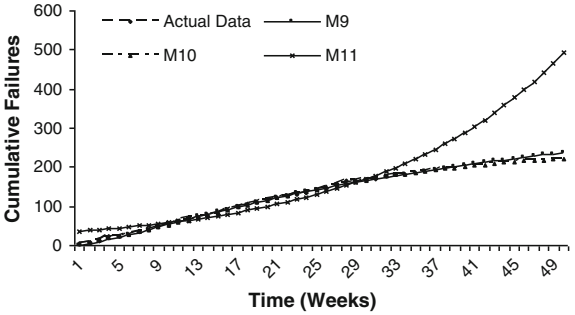


Fig. 8.4 Goodness of fit curve for testing domain based SDE models (M9–M11)

on this data set. The mean square error for this model is very high with magnitude 292.15 and R^2 value is low (0.894) as compared to the best fit model with MSE equals 6.60 and R^2 value 0.999. The goodness of fit curve for models M1–M4 is shown in Fig. 8.1, for fault complexity models in Fig. 8.2, for change point models in Fig. 8.3 and for testing domain based models in Fig. 8.4.

Exercises

1. Under what condition one should apply stochastic differential equation based SRGM.
2. Define the following
   a. Stochastic process
   b. Brownian motion
   c. $It\hat{O}$ Integrals
3. Fault detection rate per remaining fault is known to have irregular fluctuations, i.e., it is represented as $b(t) + \sigma\gamma(t)$, where $\sigma\gamma(t)$ represents a standardized Gaussian white noise. In such a case the differential equation for basic SDE model is given by

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \{b(t) + \sigma\gamma(t)\}\{a - N(t)\}.$$

Derive the solution of the above differential equation. Here $N(t)$ is a random variable which represents the number of software faults detected in the software system up to testing time t.

4. Derive the Exponential SDE based software reliability growth model. Give the expression of instantaneous and cumulative MTBF for the model.
5. Using the real life software project data given below
   a. Compute the estimates of unknown parameters of the models M1–M4, M7 and M8.
   b. Analyze and compare the results of estimation based on root mean square prediction error.
   c. Draw the graphs for the goodness of fit.

Testing time (days)	Cumulative failures	Testing time (days)	Cumulative failures
1	2	12	24
2	3	13	26
3	4	14	30
4	5	15	31
5	7	16	37
6	9	17	38
7	11	18	41
8	12	19	42
9	19	20	45
10	21	21	46
11	22		

References

1. Arnold L (1974) Stochastic differential equations. Wiley, New York
2. Wong E (1971) Stochastic processes in information and dynamical systems. McGraw-Hill, New York
3. Yamada S, Kimura M, Tanaka H, Osaki S (1994) Software reliability measurement and assessment with stochastic differential equations. IEICE Trans Fundam Electron Comput Sci E77-A(1):109–116
4. Yamada S, Nishigaki A, Kimura M (2003) A stochastic differential equation model for software reliability assessment and its goodness of fit. Int J Reliab Appl 4(1):1–11
5. Kapur PK, Anand S, Yadavalli VSS, Beichelt F (2007) A generalised software growth model using stochastic differential equation. Communication in Dependability and Quality Management Belgrade, Serbia, pp 82–96
6. Kapur PK, Anand S, Yamada S, Yadavalli VSS (2009) Stochastic differential equation-based flexible software reliability growth model. Math Prob Eng, Article ID 581383, 15 pages. doi: 10.1155/2009/581383
7. Tamura Y, Yamada S (2006) A flexible stochastic differential equation model in distributed development environment. Eur J Oper Res 168:143–152
8. Kapur PK, Singh VB, Anand S (2007) Effect of change-point on software reliability growth models using stochastic differential equation. In: 3rd international conference on reliability and safety engineering (INCRESE-2007), Udaipur, 7–19 Dec, 2007, pp 320–333

9. Kapur PK, Anand S, Yadav K (2008) Testing-domain based software reliability growth models using stochastic differential equation. In: Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems. MacMillan India Ltd, New Delhi, pp 817–830
10. Jeske DR, Zhang X, Pham L (2005) Adjusting software failure rates that are estimated from test data. IEEE Trans Reliab 54(1):107–114

Chapter 9
Discrete SRGM

9.1 Introduction

In Chap. 1, we familiarized the readers that non-homogeneous Poisson process (NHPP) based software reliability growth models (SRGM) are generally classified into two groups. The first group of models uses the execution time (i.e. CPU time) or calendar time to describe the software failure and fault removal phenomena. Such models are called continuous time models. The focus of Chaps. 2–8 was mainly on the continuous time models. Most of the research in software reliability modeling has been carried on the continuous time models. The second type of models are known as *discrete time models*, these models use the number of test cases executed as a unit for measuring the testing process [1, 2, 3]. A test case can be a single computer test run executed in second(s), minute(s), hour(s), day(s), weeks(s) or even month(s). Therefore, it includes the computer test run and length of time spent on its execution. A large number of models have been developed in the first group while fewer are there in the second group. The reason why the second group of models finds limited interest of researchers is the difficulties in terms of mathematical complexity involved in formulating and finding closed form solution of these models. In spite of all these the utility of discrete reliability growth models cannot be underestimated. Most of the observed/cited software failure data sets are discrete and as such these models many times provide better fit than their continuous time counterparts. There are only a few studies in the literature on the discrete software reliability modeling and most of the books addressing to the reliability measurement and assessment have avoided the discussion on these models. The book aims to provide the widespread knowledge to its readers on every aspect of NHPP based software reliability modeling so, this chapter is devoted entirely to the study of discrete software reliability modeling.

An NHPP based SRGM describes the failure or removal phenomenon during testing and operational phase. Using the data collected over a period of time of the ongoing testing and based on some assumption of the testing environment, one can

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_9,
© Springer-Verlag London Limited 2011

estimate the number of faults that can be removed by a specific time t and hence the reliability. The discrete counting process has already been explained in Sect. 1.5.5. Several discrete SRGM have been proposed in the literature under different set of assumptions. Here we endeavor to describe the development of discrete SRGM considering all the aspects of the testing environment that affect the testing process firstly stating the general assumption of discrete NHPP based SRGM.

9.1.1 General Assumption

1. Failure observation/fault removal phenomenon is modeled by NHPP with mean value function $m(n)$.
2. Software is subject to failures during execution caused by remaining software faults.
3. On a failure observation, an immediate effort takes place to remove the cause of failure.
4. Failure rate is equally affected by all the faults remaining in the software.

Under these general assumptions and some specific assumptions based on the testing environment different models are developed.

Notation

a	Initial fault content of the software
b	Constant fault removal rate per remaining fault per test case
$m(n)$	The expected mean number of faults removed by the nth test case
$m_f(n)$	The expected mean number of failures occurred by the nth test case
$m_r(n)$	The expected mean number of removal occurred by the nth test case
d	Constant for rate of increase in delay
p	Proportion of leading faults in the software
$m_1(t)$	Expected number of leading faults detected in the interval $(0, t]$
$m_2(t)$	Expected number of dependent faults detected in the interval $(0, t]$
n	Number of test occasions
a_i	Fault content of type i, with $\sum_{i=1}^{k} a_i = a$, where a is the total fault content
b_i	Constant failure rate/fault isolation rate per fault of type i
$b_i(n)$	Logistic learning function, i.e. fault removal rate per fault of type i
$m_{if}(n)$	Mean number of failure caused by fault-type i by n test cases
$m_{ii}(n)$	Mean number of fault-isolated of fault-type i by n test cases
$m_{ir}(n)$	Mean number of fault-removed of fault-type i by n test cases
β	Constant parameter in the logistic learning-process function

$W(n)$ The cumulative testing resources spent up to the nth test run
$w(n)$ The testing resources spent on the nth test run

9.1.2 Definition

Define

$$t = n\delta \quad \text{and} \quad \lim_{x \to 0}(1+x)^{1/x} = e \tag{9.1.1}$$

9.2 Discrete SRGM Under Perfect Debugging Environment

The early development of discrete SRGM was mainly under the perfect debugging environment. The continuous time SRGM developed under perfect debugging environment have been discussed in Chap. 2. A perfect debugging environment basically means that the testing and debugging teams are perfect in their jobs, are experienced professionals and know the detailed structure of the programs under testing. Now we describe the development of some perfect debugging SRGM in the discrete time space.

9.2.1 Discrete Exponential Model

Under the basic assumption that the expected cumulative number of faults removed between the nth and the $(n+1)$th test cases is proportional to the number of faults remaining after the execution of the nth test run, satisfies the following difference equation [4]

$$\frac{m(n+1) - m(n)}{\delta} = b(a - m(n)) \tag{9.2.1}$$

Multiplying both sides of (9.2.1) by z^n and summing over n from 0 to ∞ we get

$$\sum_{n=0}^{\infty} z^n m(n+1) - \sum_{n=0}^{\infty} z^n m(n) = ab\delta \sum_{n=0}^{\infty} z^n - b\delta \sum_{n=0}^{\infty} z^n m(n)$$

Solving the above difference equation under the initial condition $m(n=0) = 0$ and using Probability Generating Function (PGF) given as

$$P(z) = \sum_{n=0}^{\infty} z^n m(n) \tag{9.2.2}$$

We get the mean value function of the exponential SRGM

$$m(n) = a(1 - (1 - b\delta)^n) \tag{9.2.3}$$

The model describes an exponential failure growth curve. The equivalent continuous SRGM corresponding to (9.2.3) is obtained taking limit $\delta \to 0$ and using the definition (9.1.1), i.e.

$$m(n) = a(1 - (1 - b\delta)^n) \to a(1 - e^{-bt}) \text{ as } \delta \to 0 \tag{9.2.4}$$

Goel and Okumoto [5] model is the continuous equivalent model of the above discrete exponential model. It may be noted here the continuous counterpart of most of the SRGM discussed in this chapter is discussed in the previous chapters, which can be obtained from their discrete versions following the procedure as above, i.e. taking limit $\delta \to 0$ and using the definition (9.1.1).

9.2.2 Modified Discrete Exponential Model

Assuming that the software contains two types of errors [4] type I and type II, we can write the difference equation corresponding to faults of each type as

$$\frac{m_1(n+1) - m_1(n)}{\delta} = b_1(a_1 - m_1(n)) \tag{9.2.5}$$

and

$$\frac{m_2(n+1) - m_2(n)}{\delta} = b_2(a_2 - m_2(n)) \tag{9.2.6}$$

where $a = a_1 + a_2$ and $b_1 > b_2$.

Solving the above equation by the method of PGF as above we get the mean value function for the SRGM in discrete time space.

$$m_1(n) = a_1(1 - (1 - b_1\delta)^n)$$

$$m_2(n) = a_2(1 - (1 - b_2\delta)^n)$$

$$m(n) = m_1(n) + m_2(n) = \sum_{i=1}^{2} a_i(1 - (1 - b_i\delta)^n) \tag{9.2.7}$$

The equivalent continuous SRGM corresponding to (9.2.7) is obtained taking limit $\delta \to 0$.

$$m(n) = \sum_{i=1}^{2} a_i(1 - (1 - b_i\delta)^n) \to \sum_{i=1}^{2} a_i(1 - e^{-b_i t}) \text{ as } \delta \to 0 \tag{9.2.8}$$

The continuous equivalent model is proposed by [4]. The models discussed in this section describe an exponential curve. In a number of practical applications exponential models are used. The main reason for this is due to their simple mathematical forms and less number of unknown parameters. However we know that exponential models account to a uniform operational profile, which seems to be unrealistic in many practical applications. It led to the development of S-shaped and flexible models as they can well describe the non-uniform environment. In the following sections we describe some of the S-shaped and flexible models in discrete times.

9.2.3 Discrete Delayed S-Shaped Model

This model [6] describes the debugging process as a two-stage process—first, on the execution of a test case a failure is observed and second, on a failure the corresponding fault is removed. Accordingly, following the general assumptions of a discrete SRGM the testing process is modeled by the following difference equations

$$\frac{m_f(n+1) - m_f(n)}{\delta} = b(a - m_f(n)) \tag{9.2.9}$$

and

$$\frac{m_r(n+1) - m_r(n)}{\delta} = b(m_f(n+1) - m_r(n)) \tag{9.2.10}$$

Solving (9.2.10) by the method of PGF and initial condition $m_f(n = 0) = 0$, we get

$$m_f(n) = a(1 - (1 - b\delta)^n) \tag{9.2.11}$$

Substituting value of $m_f\,(n + 1)$ from (9.2.11) in (9.2.10) and solving by the method of PGF with initial condition $m_r(n = 0) = 0$, we get

$$m_r(n) = a[1 - (1 + bn\delta)(1 - b\delta)^n] \tag{9.2.12}$$

The equivalent continuous SRGM corresponding to (9.2.12), is obtained taking limit $\delta \to 0$, i.e.

$$m_r(n) = a[1 - (1 + bn\delta)(1 - b\delta)^n] \to a(1 - (1 + bt)e^{-bt}) \tag{9.2.13}$$

The continuous model is due to [7] and describes the delayed fault removal phenomenon.

9.2.4 Discrete SRGM with Logistic Learning Function

The SRGM discussed above assume a constant rate of fault removal per remaining error. However in practical situation as the testing goes on the experience of the testing team increases with the software under testing and therefore it is expected that fault removal rate per remaining error will follow a logistic learning function. The model discussed in this section [8] incorporates the learning process of testing team into the SRGM. The difference equation for the model is given as

$$\frac{m(n+1)-m(n)}{\delta} = \frac{b}{1+\beta(1-b\delta)^{n+1}}(a-m(n)) \tag{9.2.14}$$

The mean value function corresponding to the above difference equation with the initial condition $m(n=0)=0$ is

$$m(n) = \frac{a}{1+\beta(1-b\delta)^{n}}[1-(1-b\delta)^{n}] \tag{9.2.15}$$

The equivalent continuous SRGM corresponding to above discrete SRGM is obtained taking limit $\delta \to 0$, i.e.

$$m(n) = \frac{a}{1+\beta(1-b\delta)^{n}}[1-(1-b\delta)^{n}] \to \frac{a}{1+\beta e^{-bt}}\left(1-e^{-bt}\right) \tag{9.2.16}$$

9.2.5 Modeling Fault Dependency

The test team can remove some additional faults in the software, without these faults causing any failure during the removal of identified faults, although this may involve some additional effort. However, removal of these faults saves the testing time in terms of their removal with failure observation. Faults, which are removed consequent to a failure, are known as a leading fault whereas the additional faults removed, which may have caused failure in future are known as dependent faults. In this section we develop some of the models in this category in discrete time space.

9.2.5.1 Discrete SRGM for Error Removal Phenomenon

In addition to considering underling fault dependency this model also describes the debugging time lag-after failure observation [9]. In the previous chapters we explained the need of modeling fault detection and fault removal processes separately. In general the assumption of immediate removal of faults on the detection of a failure does not hold true. Usually there is a time lag in the removal process

after the detection process. The removal time indeed can also be not negligible, due to its dependence on a number of factors such as complexity of the detected faults, skills of the debugging team, available manpower, software development environment, etc. Hence in general testing environment fault removal may take a longer time after detection.

Under the assumption that while removing leading faults the testing team may remove some dependent faults, the difference equation for the fault removal process can be written as

$$\frac{m_r(n+1)-m_r(n)}{\delta} = p[a - m_r(n)] + \frac{q}{a} m_r(n+1)[a - m_r(n)] \tag{9.2.17}$$

where q and p are the rates of leading and dependent fault detection, respectively.

Solving (9.2.17) by the method of PGF and initial condition $m(n = 0) = 0$, we obtain the mean value function of a flexible SRGM, given as

$$m_r(n) = a\left[\frac{1-\{1-\delta(p+q)\}^n}{1+(q/p)\{1-\delta(p+q)\}^n}\right] \tag{9.2.18}$$

The equivalent continuous SRGM [10] corresponding to (9.2.18), is obtained taking limit $\delta \to 0$, i.e.

$$m_r(n) = a\left[\frac{1-\{1-(p+q)\}^n}{1+(q/p)\{1-(p+q)\}^n}\right] \to m\left[\frac{1-e^{-(p+q)t}}{1+(q/p)e^{-(p+q)t}}\right] \text{ as } \delta \to 0 \tag{9.2.19}$$

9.2.5.2 Discrete Time Fault Dependency with Lag Function

This model is based on the assumption that there exists definite time lag between the detection of leading faults and the corresponding dependent faults [11]. Assuming that the intensity of dependent fault detection is proportional to the number of dependent faults remaining in the software and the ratio of leading faults removed to the total leading faults, the difference equation for leading faults is given as

$$m_1(n+1) - m_1(n) = b[ap - m_1(n)] \tag{9.2.20}$$

The leading faults with the initial condition $m_1(n = 0) = 0$ are described by the mean value function

$$m_1(n) = a[1-(1-b)^n] \tag{9.2.21}$$

The dependent fault detection can be put as the following differential equation

$$m_2(n+1) - m_2(n) = c[a(1-p) - m_2(n)]\frac{m_1(n+1-\Delta n)}{ap}, \tag{9.2.22}$$

where Δn is the lag depending upon the number of test occasions.

When $\Delta n = \log_{(1-b)^{-1}}(1+dn)$, we get $m_2(n)$ under the initial condition $m_2(n=0)=0$ as

$$m_2(n) = a(1-p)\left[1 - \prod_{i=1}^{n}\left\{1 - c\left(1-(1-b)^i(1+(i-1)d\right)\right\}\right] \tag{9.2.23}$$

Hence, the expected total number of faults removed in n test occasion is

$$m(n) = a\left[1 - p(1-b)^n + (1-p)\prod_{i=1}^{n}\left\{1 - c\left(1-(1-b)^i(1+(i-1)d\right)\right\}\right] \tag{9.2.24}$$

The equivalent continuous SRGM corresponding to above discrete SRGM is

$$m(t) = a\left[1 - pe^{-bt} - (1-p)e^{-cf(t)}\right], \tag{9.2.25}$$

where $f(t) = t + \frac{1}{b}\left(1+\frac{d}{b}\right)\left(e^{-bt}-1\right) + \frac{d}{b}te^{-bt}$.

9.3 Discrete SRGM under Imperfect Debugging Environment

A number of imperfect debugging continuous time SRGM have been discussed in Chap. 3. Considering the imperfect debugging phenomena in reliability modeling is very important to the reliability measurement as it is related to the efficiency of the testing and debugging teams. Besides this consideration, helps the developers in having an insight the measure of testing efficiency, which they can use to formulate the testing teams and strategies and make decisions related to the changes in the testing strategies and team compositions required to pace up the testing at any stage. The study of imperfect debugging environment is very limited in discrete time space owning to the complexity of exact form solution of the mean value function. In this section we discuss a discrete SRGM with two types of imperfect debugging namely—imperfect fault debugging and error generation. The difference equation for a discrete SRGM under imperfect debugging environment incorporating two types of imperfect debugging and learning process of the testing team as testing progresses is given by [12]

$$\frac{m_r(n+1) - m_r(n)}{\delta} = b(n+1)(a(n) - m_r(n)). \tag{9.3.1}$$

Let us define

$$a(n) = a_0(1+\alpha\delta)^n \tag{9.3.2}$$

$$b(n+1) = \frac{b_0 p}{1 + \beta(1 - b_0 p\delta)^{n+1}} \tag{9.3.3}$$

An increasing $a(n)$ implies an increasing total number of faults, and thus reflects fault generation. Whereas, $b(n+1)$ is a logistic learning function representing the learning of the testing team and is affected by the probability of fault removal on a failure.

Substituting the above forms of $a(n)$ and $b(n+1)$ in the difference equation (9.3.1) and solving by the method of PGF, the closed form solution is as given below

$$m_r(n) = \frac{a_0 b_0 p\delta}{1 + \beta(1 - b_0 p\delta)^n}\left[\frac{(1+\alpha\delta)^n - (1 - b_0 p\delta)^n}{(\alpha\delta + b_0 p\delta)}\right], \tag{9.3.4}$$

where $m_r(n=0) = 0$.

If the imperfect fault debugging parameter $p = 1$ and fault generation rate $\alpha = 0$, i.e. the testing process is perfect, then mean value function of the removal process, $m_r(n)$ given by expression (9.3.4) reduces to

$$m_r(n) = a_0\left[\frac{1 - (1 - b_0\delta)^n}{1 + \beta(1 - b_0\delta)^n}\right] \tag{9.3.5}$$

which is perfect debugging flexible discrete SRGM (9.2.19) with $b_0 = p + q$ and $\beta = q/p$. The equivalent continuous SRGM corresponding to (9.3.4) is obtained taking limit $\delta \to 0$, i.e.

$$\frac{a_0 b_0 p\delta}{1 + \beta(1 - b_0 p\delta)^n}\left[\frac{(1+\alpha\delta)^n - (1 - b_0 p\delta)^n}{(\alpha\delta + b_0 p\delta)}\right] \to \frac{a_0 b_0 p}{1 + \beta e^{-b_0 pt}}\left[\frac{e^{\alpha t} - e^{-b_0 pt}}{\alpha + b_0 p}\right] \tag{9.3.6}$$

The imperfect debugging discrete SRGM discussed above is a flexible model, as it possesses the properties of exponential as well as s-shaped models.

9.4 Discrete SRGM with Testing Effort

Failure observation, fault identification and removal are dependent upon the nature and amount of testing efforts spent on testing. The time dependent behavior of the testing effort has been studied by many researchers in the literature (refer to Sect. 2.6) but most of the work relates the testing resources to the testing time. A discrete test effort function describes the distribution or consumption pattern of testing resources with respect to the executed test cases. Here we discuss a discrete SRGM with testing effort, assuming that the cumulative testing resources spent up to the nth test run, $W(n)$, is described by a discrete Rayleigh curve, i.e.

$$w(n+1) = W(n+1) - W(n) = \beta(n+1)[\alpha - W(n)] \tag{9.4.1}$$

Solving (9.4.1) following PGF method we get

$$W(n) = \alpha\left(1 - \prod_{i=0}^{n}(1 - i\beta)\right) \tag{9.4.2}$$

and hence

$$w(n) = \alpha\beta n \prod_{i=0}^{n-1}(1 - i\beta) \tag{9.4.3}$$

Under the above assumptions, the difference equation for an exponential SRGM is written as

$$\frac{m(n+1) - m(n)}{w(n)} = b(a - m(n)) \tag{9.4.4}$$

Mean value function corresponding to the above difference equation is

$$m(n) = a\left(1 - \prod_{i=0}^{n}(1 - bw(i))\right) \tag{9.4.5}$$

This model is due to [13].

9.5 Modeling Faults of Different Severity

SRGM which categorise the faults based on the complexity of fault detection and removal process, provides in most cases very accurate estimation and prediction of the reliability measures. Complexity of faults is considered in terms of the delay occurring in the removal process after the failure observation. More complex the fault more is delay in the fault isolation and removal after the failure observation. Various continuous time fault complexity based SRGM have been discussed in the previous chapters under the varying sets of assumptions and considering different aspects of software testing process and the factors that influence the reliability growth. In the next section we discuss the models conceptualizing the concept of faults of different complexity in the discrete time space.

9.5.1 Generalized Discrete Erlang SRGM

Assuming that the software consists of n different types of faults and on each type of fault a different strategy is required to remove the cause of failure due to that fault, we assume that for a type i (i = I, II,..., k) fault, i different processes (stages)

are required to remove the cause of failure. Accordingly we may write the following difference equations for faults of each type [1].

9.5.1.1 Modeling Simple Faults (Fault-Type I)

The simple fault removal is modeled as a one-stage process

$$m_{11}(n+1) - m_{11}(n) = b_1(a_1 - m_{11}(n)) \tag{9.5.1}$$

9.5.1.2 Modeling the Hard Faults (Fault-Type II)

The harder type of faults is assumed to take more testing effort. The removal process for such faults is modeled as a two-stage process.

$$\begin{aligned} m_{21}(n+1) - m_{21}(n) &= b_2(a_2 - m_{21}(n)) \\ m_{22}(n+1) - m_{22}(n) &= b_2(m_{21}(n+1) - m_{22}(n)) \end{aligned} \tag{9.5.2}$$

9.5.1.3 Modeling the Fault-Type k

The modeling procedure of the hard fault can be extended to formulate a model that describes the removal of a fault-type k with k stages in removal.

$$\begin{aligned} m_{k1}(n+1) - m_{k1}(n) &= b_k(a_k - m_{k1}(n)) \\ m_{k2}(n+1) - m_{k2}(n) &= b_k(m_{k1}(n+1) - m_{k2}(n)) \\ &\cdots \\ m_{kk}(n+1) - m_{kk}(n) &= b_k(m_{k,k-1}(n+1) - m_{kk}(n)) \end{aligned} \tag{9.5.3}$$

Here $m_{ij}(.)$ represent the mean value function for the ith type of fault in the jth stage. Solving the above difference equations, we get the general solution for the mean value function for the removal process for each type of fault

$$m_i(n) = m_{ii}(n) = a_i(1 - (1 - b_i)^n \left(\sum_{j=0}^{i-1} \frac{b_i^j}{j!(n+j)} \prod_{l=0}^{j} (n+l) \right) \quad i = 1, \ldots, k \tag{9.5.4}$$

Since $m(n) = \sum_{i=1}^{k} m_i(n)$, we get

$$m(n) = \sum_{i=1}^{k} a_i(1 - (1 - b_i)^n) \left(\sum_{j=0}^{i-1} \frac{b_i^j}{j!(n+j)} \prod_{l=0}^{j} (n+l) \right) \tag{9.5.5}$$

In particular, we have

$$m_1(n) = m_{11}(n) = a_1(1 - (1 - b_1)^n)$$

$$m_2(n) = m_{22}(n) = a_2(1 - (1 + b_2 n)(1 - b_2)^n)$$

and

$$m_3(n) = m_{33}(n) = a_3\big(1 - \big(\big(1 + b_3 n + \big(b_3^2 n(n+1)/2\big)\big)\big)(1 - b_3)^n\big) \tag{9.5.6}$$

The removal rate per fault for the above three types of faults is given as

$$d_1(n) = b,\ d_2(n) = \frac{b_2^2(n+1)}{b_2 n + 1} \quad \text{and} \quad d_3(n) = \frac{b_3^3(n^2 + 3n + 2)}{2\big(\big(b_3^2 n(n+1)/2\big) + b_3 n + 1\big)},$$

respectively. We observe that $d_1(n)$ is constant with respect to n_1 while $d_2(n)$ and $d_3(n)$ increase with n and tend to b_2 and b_3 as $n \to \infty$. Thus in the steady state, $m_2(n)$and $m_3(n)$ behave similarly as $m_1(n)$ and hence there is no loss of generality in assuming steady state rates b_2 and b_3 equal to b_1. Generalizing for arbitrary k, we can assume $b_1 = b_2 = \ldots = b_k = b$(say). We thus have

$$m_i(n) \equiv m_{ii}(n) = a_i(1 - (1 - b)^n \left(\sum_{j=0}^{i-1} \frac{b^j}{j!(n+j)} \prod_{l=0}^{j} (n + l)\right), \tag{9.5.7}$$

and

$$m(n) = \sum_{i=1}^{k} a_i(1 - (1 - b)^n) \sum_{i=1}^{k} \left(\sum_{j=0}^{i-1} \frac{b^j}{j!(n+j)} \prod_{l=0}^{j} (n + l)\right) \tag{9.5.8}$$

The equivalent continuous time model [14], modeling errors of different severity is

$$m(t) = \sum_{i=1}^{k} a_i \left[1 - e^{-b_i t} \left(\sum_{j=0}^{i-1} \frac{(b_i t)^j}{j!}\right)\right] \tag{9.5.9}$$

which can be derived as a limiting case of discrete model substituting $t = n\delta$ and taking limit $\delta \to 0$.

9.5.2 *Discrete SRGM with Errors of Different Severity Incorporating Logistic Learning Function*

Kapur et al. [15] incorporated a logistic learning function during the removal phase, for capturing variability in the growth curves depending on software test

conditions and learning process of the test team as the number of test runs executed increases for modeling errors of different severity in the above model. Such framework is very much suited for object-oriented programming and distributed development environments. Assuming software contains finite number of fault types and the time delay between the failure observations and its subsequent removal represents the severity of the faults, the concept of errors of different severity with logistic rate of fault removal per remaining fault can be modeled as follows

9.5.2.1 Modeling the Simple Faults (Fault-Type I)

The simple fault removal process is modeled as a one-stage process

$$m_{1r}(n+1) - m_{1r}(n) = b_1(n+1)(a_1 - m_{1r}(n)), \tag{9.5.10}$$

where $b_1(n+1) = b_1$.

Solving the above difference equation using the PGF with the initial condition $m_{1r}(n=0) = 0$, we get

$$m_{1r}(n) = a_1(1 - (1 - b_1)^n) \tag{9.5.11}$$

9.5.2.2 Modeling the Hard Faults (Fault-Type II)

The harder type of faults is assumed to take more testing-effort. The removal process for such faults is modeled as a two-stage process,

$$m_{2f}(n+1) - m_{2f}(n) = b_2(a_2 - m_{2f}(n)) \tag{9.5.12}$$

$$m_{2r}(n+1) - m_{2r}(n) = b_2(n+1)(m_{2f}(n+1) - m_{2r}(n)) \tag{9.5.13}$$

where $b_2(n+1) = \frac{b_2}{1+\beta(1-b_2)^{n+1}}$.

Solving the above system of difference equations using the PGF with the initial conditions $m_{2f}(n=0) = 0$ and $m_{2r}(n=0) = 0$ we get

$$m_{2r}(n) = a_2 \frac{1 - (1 + b_2 n)(1 - b_2)^n}{1 + \beta(1 - b_2)^n} \tag{9.5.14}$$

9.5.2.3 Modeling the Complex Faults (i.e. Fault-Type III)

The complex fault removal process is modeled as a three-stage process,

$$m_{3f}(n+1) - m_{3f}(n) = b_3(a_3 - m_{3f}(n)) \tag{9.5.15}$$

$$m_{3i}(n+1) - m_{3i}(n) = b_3(m_{3f}(n+1) - m_{3i}(n)) \tag{9.5.16}$$

$$m_{3r}(n+1) - m_{3r}(n) = b_3(n+1)(m_{3i}(n+1) - m_{3r}(n)), \tag{9.5.17}$$

where $b_3(n+1) = \frac{b_3}{1+\beta(1-b_3)^{n+1}}$.

Solving the above system of difference equations using the PGF with the initial conditions $m_{3f}(n=0) = 0$, $m_{3i}(n=0) = 0$ and $m_{3r}(n=0) = 0$, we get

$$m_{3r}(n) = a_3 \frac{1 - (1 - b_3 n + \frac{b_3^2 n(n+1)}{2})(1-b_3)^n}{1 + \beta(1-b_3)^n} \tag{9.5.18}$$

9.5.2.4 Modeling the Fault-Type k

The modeling procedure of the complex fault can be extended to formulate a model that describes the removal of a fault-type k with r stages (r can be equal to k) of removal.

$$m_{kf}(n+1) - m_{kf}(n) = b_k(a_k - m_{kf}(n)) \tag{9.5.19}$$

$$m_{kq}(n+1) - m_{kq}(n) = b_k(m_{kf}(n+1) - m_{kq}(n)) \tag{9.5.20}$$

$$\cdots$$

$$m_{kr}(n+1) - m_{kr}(n) = b_k(n+1)(m_{k(r-1)}(n+1) - m_{kr}(n)), \tag{9.5.21}$$

where $b_k(n+1) = \frac{b_k}{1+\beta(1-b_k)^{n+1}}$.

Solving the above system of difference equations using the PGF with the initial conditions, $m_{kf}(n=0) = m_{kf}(n=0) = ,\ldots, m_{kr}(n=0) = 0$, we get

$$m_{kr}(n) = a_k \frac{1 - \left(1 + \sum_{j=1}^{k-1} \frac{b_k^j}{j!(n+j)} \prod_{l=0}^{j} (n+l)\right)(1-b_k)^n}{(1 + \beta(1-b_k)^n)} \tag{9.5.22}$$

9.5.2.5 Modeling the Total Fault Removal Phenomenon

The total fault removal phenomenon is the superposition of the NHPP with mean value functions given in Eqs. (9.5.11), (9.5.14), (9.5.18) and (9.5.22). Thus, the mean value function of the SRGM is

$$m(n) = \sum_{i=1}^{k} m_{ir}(n) = a_i(1-(1-b_i)^n) + \sum_{i=2}^{k} a_i \frac{1-\left(1+\sum_{j=1}^{i-1}\frac{b_i^j}{j!(n+j)}\prod_{l=0}^{j}(n+l)\right)(1-b_i)^n}{(1+\beta(1-b_i)^n)} \tag{9.5.23}$$

where $m(n)$ provides the general framework with k types of faults.

The fault removal rate per fault for fault-types, 2 and 3 are given, respectively, as follows

$$d_1(n) = \frac{m_1(n+1)-m_1(n)}{a_i - m_1(n)} = b_1 \tag{9.5.24}$$

$$d_2(n) = \frac{m_2(n+1)-m_2(n)}{a_2 - m_1(n)} = \frac{b_2(1+\beta+b_2 n)-b_2(1+\beta(1-b_2)^n)}{(1+\beta(1-b_2)^n)(1+\beta+b_2 n)} \tag{9.5.25}$$

$$d_3(n) = \frac{b_3(1+\beta+b_3 n+\frac{b_3^2 n(n+1)}{2})-b_3(1+\beta(1-b_3)^n)(1+b_3 n)}{(1+\beta(1-b_3)^n)(1+\beta+b_3 n+\frac{b_3^2 n(n+1)}{2})} \tag{9.5.26}$$

It is observed that $d_1(n)$ is constant with respect to n while $d_2(n)$ and $d_3(n)$ increase monotonically with n and tend to constants b_2 and b_3 as $n \to \infty$. Thus, in the steady state, $m_{2r}(n)$ and $m_{3r}(n)$ behave similarly as $m_{1r}(n)$ and hence without loss of generality we can assume the steady state rates b_2 and b_3 to be equal to b_1. After substituting $b_2 = b_3 = b_1$ in the right-hand side of Eqs. (9.5.25) and (9.5.26), one can see that $b_1 > d_2(n) > d_3(n)$, which is in accordance with the severity of the faults. Generalizing for arbitrary k, assuming $b_1 = b_2 = \ldots = b_k = b$ (say) we may write (9.5.23) as follows

$$m(n) = \sum_{i=1}^{n} m_{ir}(n) = a_1(1-(1-b)^n) + \sum_{i=2}^{k} a_i \frac{1-\left(1+\sum_{j=1}^{i-1}\frac{b^j}{j!(n+j)}\prod_{l=1}^{j}(n+l)\right)(1-b)^n}{(1+\beta(1-b)^n)} \tag{9.5.27}$$

The equivalent continuous time model [16], modeling errors of different severity is

$$m(t) = \sum_{i=1}^{n} m_{ir}(t) = a_1\left(1-e^{-bt}\right) + \sum_{i=2}^{k} a_i \frac{1-\left(\sum_{j=0}^{k-1}\frac{(bt)^j}{j!}\right)e^{-bt}}{(1+\beta e^{-bt})} \tag{9.5.28}$$

which can be derived as a limiting case of discrete model substituting $t = n\delta$ and taking limit $\delta \to 0$.

9.5.3 Discrete SRGM Modeling Severity of Faults with Respect to Test Case Execution Number

In the fault complexity based software reliability growth modeling faults can be categorized on the basis of their time to detection. During the early stage of testing the faults are easily detectable and can be called simple faults or trivial faults. As the complexity of faults increases, so does their detection time. Faults, which take maximum time for detection, are termed as complex faults.

For classification of faults on the basis of their detection times [17], first we define non-cumulative instantaneous error detection function $f(n)$ using discrete SRGM for error removal phenomenon discussed in Sect. 9.2.5, which is given by first-order difference equation of $m(n)$.

$$\begin{aligned} f(n) = \Delta m(n) &= \frac{m(n+1) - m(n)}{\delta} \\ &= \frac{\overline{N}p(p+q)^2[1-\delta(p+q)]^n}{[p+q(1-\delta(p+q))^n]\left[p+q(1-\delta(p+q))^{n+1}\right]} \end{aligned} \tag{9.5.29}$$

Above, $f(n)$ defines the mass function for non-cumulative fault detection. It takes the form of a bell-shaped curve and it represents the rate of fault removal for n. Peak of $f(n)$ occurs when

$$n = \left\{ \begin{array}{ll} [n^*] & \text{if } f([n^*]) \geq f([n^*]+1) \\ [n^*]+1 & \text{otherwise} \end{array} \right) \tag{9.5.30}$$

where $n^* = \frac{\log(p/q)}{\log(1-\delta(p+q))} - 1$ and $[n^*] = \{n : \max(n \leq n^*), n \in Z\}$.

Then as $\delta \to 0$, i.e. n^* converges to inflection point of continuous s-shaped SRGM due to [10]. Using $t = n\delta$ we get

$$t^* = \frac{\delta \log(p/q)}{\log(1-\delta(p+q))} - \delta \to -\frac{\log(p/q)}{p+q} \text{ as } \delta \to 0$$

The corresponding $f(n^*)$ is given by

$$f(n^*) = \frac{\overline{N}(p+q)^2}{2q(2-\delta(p+q))} \to f(t^*) = \frac{\overline{N}(p+q)^2}{4q} \text{ as } \delta \to 0$$

The curve for $f(n)$, the non-cumulative error detection is symmetric about point n^* up to $2n^*+1$. Here $f(0) = f(2n^*+1) = \overline{N}p/(1-\delta q)/$. As $\delta \to 0$ is symmetric about t^* up to $2t^*$ then, $f(t=0) = f\left(2t^*\right) = \overline{N}p$. To get the insight into type of trend shown by $f(n)$, we need to find $\Delta f(n)$, i.e. rate of change in non-cumulative error detection $f(n)$.

$$\Delta f(n) = \frac{f(n+1) - f(n)}{\delta}$$

$$\Delta f(n) = \frac{-\overline{N}p(p+q)^3[1-\delta(p+q)]^n\left[p - q(1-\delta(p+q))^{n+1}\right]}{[p+q(1-\delta(p+q))^n]\left[p+q(1-\delta(p+q))^{n+1}\right].\left[p+q(1-\delta(p+q))^{n+2}\right]} \tag{9.5.31}$$

The trend shown by $f(n)$ can be summarized as in Table 9.1 and the size of each fault category is shown in Table 9.2.

Here we observe that error removal rate increases for (0, n_1^*) with increasing rate and decreasing rate for $(n_1^* + 1$ to $n^*)$. It is because of the fact that as the testing grows so does the skill of the testing team. The faults detected during (0, n_1^*) are relatively easy faults while those detected during $(n_1^* + 1,\ n^*)$ are relatively difficult errors. The n^* is point of maxima for $f(n)$. For $(n_1^* + 1,\ n_2^*)$, the error detection rate decreases, i.e. lesser number of errors are detected upon failure. These errors can be defined as relatively hard errors. For $(n_2^* + 1,\ \infty)$, very few errors are detected upon failure. So testing is terminated. Errors detected beyond $n_2^* + 1$ are relatively complex errors. Here, $n_1^*$and n_2^* are points of inflection for $f(n)$.

Point of maxima of $\Delta f(n)$

$$n_1^* = \begin{cases} [n_1] & \text{if } \Delta f[n_1]) \geq \Delta f([n_1]+1) \\ [n_1]+1 & \text{otherwise} \end{cases}, \tag{9.5.32}$$

where $n_1 = \frac{1}{\log(1-\delta(p+q))} \log\left[\frac{p}{q}\left\{\frac{(2-\delta(p+q))+\sqrt{(1-\delta(p+q))^2+(2-\delta(p+q))}}{(1-\delta(p+q))}\right\}\right] - 1$

and $[n_1] = \{n | \max(n \leq n_1), n \in Z\}$.

Point of minima of $\Delta f(n)$

$$n_2^* = \begin{cases} [n_2] & \text{if } \Delta f[n_2]) \geq \Delta f([n_2]+1) \\ [n_2]+1 & \text{otherwise} \end{cases}, \tag{9.5.33}$$

Table 9.1 Trends in rate of change in non-cumulative error detection

No. of test cases	Trends in $f(n)$
Zero to n_1^*	Increasing at an increasing rate
$n_1^* + 1$ to n^*	Increasing at a decreasing rate
$n^* + 1$ to n_2^*	Decreasing at an increasing rate
$n_2^* + 1$ to ∞	Decreasing at a decreasing rate

Table 9.2 Size of different fault categories

No. of test cases	Fault category	Expression for the fault category size
Zero to n_1^*	Easy faults	$m(n_1)$
$n_1^* + 1$ to n^*	Difficult fault	$m(n^*)$–$m(n_1)$
$n^* + 1$ to n_2^*	Hard faults	$m(n_2)$–$m(n^*)$
Beyond n_2^*	Complex fault	$\bar{N} - m(n_2)$

$$\text{where } n_2 = \frac{1}{\log(1-\delta(p+q))} \log \left[\frac{p}{q} \left\{ \frac{(2-\delta(p+q)) - \sqrt{(1-(\overline{p+q})\delta)^2 + (2-\delta(p+q))}}{(1-\delta(p+q))} \right\} \right] - 1$$

and $[n_2] = \{n | \max(n \leq n_2), n \in \mathrm{Z}\}$.

It may be noted that the corresponding inflection points T_1 and T_2, for the continuous case can be derived from n_1 and n_2 as $\delta \to 0$, i.e.

$$n_1 \to T_1 = -\frac{1}{p+q} \log \left[\frac{p}{q} \left(2 - \sqrt{3} \right) \right] \text{ as } \delta \to 0$$

$$n_2 \to T_2 = -\frac{1}{p+q} \log \left[\frac{p}{q} \left(2 + \sqrt{3} \right) \right] \text{ as } \delta \to 0$$

9.6 Discrete Software Reliability Growth Models for Distributed Systems

We are aware that computing systems has reached the state of distributed computing which is built on the following three components: (a) personal computers, (b) local and fast wide area networks, and (c) system and application software. By amalgamating computers and networks into one single computing system and providing appropriate system software, a distributed computing system has created the possibility of sharing information and peripheral resources. Furthermore, these systems improved performance of a computing system and individual users. Distributed computing systems are also characterized by enhanced availability, and increased reliability. A distributed development project with some or all of the software components generated by different teams presents complex issues of quality and reliability of the software.

The SRGM for distributed development environment discussed in this section [18] considers that software system consists of finite number of reused and newly developed components and takes into account the time lag between the failure and fault isolation/removal processes for the newly developed components. Faults in the reused components are assumed to be of simple types and an exponential

failure curve is suggested to describe the failure/fault removal phenomenon. S-shaped failure curves are used to describe the failure and fault removal phenomena of hard and complex faults present in the newly developed components. The fault removal rate for these components is described by a discrete logistic learning function as it is expected the learning process will grow with time.

Additional Notation

a_i	Initial fault content of type i reused component
a_j	Initial fault content of type j newly developed components with hard faults
a_k	Initial fault content of type k newly developed component with complex faults
b_i	Proportionality constant failure rate per fault of ith reused component
b_j	Proportionality constant failure rate per fault of jth newly developed component
b_k	Proportionality constant failure rate per fault of kth newly developed component
$b_j(n)$	Fault removal rate per fault of jth newly developed component
$b_k(n)$	Fault removal rate per fault of kth newly developed component
$m_{ir}(n)$	Mean number of faults removed from ith reused component by n test cases
$m_{jf}(n)$	Mean number of failures caused by jth newly developed component by n test cases
$m_{jr}(n)$	Mean faults removal number from jth newly developed component by n test cases
$m_{kf}(n)$	Mean failures number caused by kth newly developed component by n test cases
$m_{ku}(n)$	Mean faults isolation number of kth newly developed component by n test cases
$m_{kr}(n)$	Mean faults removal number from kth newly developed component by n test cases.

9.6.1 Modeling the Fault Removal of Reused Components

9.6.1.1 Modeling Simple Faults

Fault removal process of reused components is modeled as one-stage processes

$$\frac{m_{ir}(n+1) - m_{ir}(n)}{\delta} = b_i(n+1)(a_i - m_{ir}(n)),$$

where

$$b_i(n+1) = b_i \tag{9.6.1}$$

Solving the above difference equation using PGF method with the initial condition $m_{ir}(n=0) = 0$, we get

$$m_{ir}(n) = a_i(1-(1-\delta b_i)^n) \tag{9.6.2}$$

9.6.2 Modeling the Fault Removal of Newly Developed Components

Software faults in the newly developed software component are assumed to be of hard or complex types. Time required for fault removal depends on the complexity of isolation and removal processes. The removal phenomenon of these faults is modeled as two-stage or three-stage process according to the time lag for removal.

9.6.2.1 Components Containing Hard Faults

The removal process for hard faults is modeled as a two-stage process, given as

$$\frac{m_{jf}(n+1)-m_{jf}(n)}{\delta} = b_j\big(a_j - m_{jf}(n)\big) \tag{9.6.3}$$

$$\frac{m_{jr}(n+1)-m_{jr}(n)}{\delta} = b_j(n+1)\big(m_{jf}(n+1) - m_{jr}(n)\big), \tag{9.6.4}$$

where

$$b_j(n+1) = \frac{b_j}{1+\beta(1-b_j)^{n+1}}$$

Solving the above system of difference equations using PGF method with the initial conditions $m_{if}(n=0) = 0$ and $m_{jr}(n=0) = 0$, we get

$$m_{jr}(n) = a_j\frac{1-\big(1+\delta b_j n\big)\big(1-\delta b_j\big)^n}{1+\beta\big(1-b_j\big)^n} \tag{9.6.5}$$

9.6.2.2 Components Containing Complex Faults

There can be components having more complex faults. These faults can require more effort for removal after isolation. Hence they need to be modeled with greater

time lag between failure observation and removal. The third stage added below to the model for hard faults serves the purpose.

$$\frac{m_{kf}(n+1)-m_{kf}(n)}{\delta}=b_k\big(a_k-m_{kf}(n)\big) \tag{9.6.6}$$

$$\frac{m_{ku}(n+1)-m_{ku}(n)}{\delta}=b_k\big(m_{kf}(n+1)-m_{ku}(n)\big) \tag{9.6.7}$$

$$\frac{m_{kr}(n+1)-m_{kr}(n)}{\delta}=b_k(n+1)(m_{ku}(n+1)-m_{kr}(n)), \tag{9.6.8}$$

where

$$b_k(n+1)=\frac{b_k}{1+\beta(1-b_k)^{n+1}}$$

Solving the above system of difference equations using PGF method with the initial conditions $m_{kf}(n=0)=0$, $m_{ku}(n=0)=0$ and $m_{kr}(n=0)=0$, we get

$$m_{kr}(n)=a_3\frac{1-\big(1+b_kn\delta+\big(b_k^2n\delta(n+1)\delta\big)/2\big)(1-\delta b_k)^n}{1+\beta(1-b_k)^n} \tag{9.6.9}$$

9.6.3 Modeling Total Fault Removal Phenomenon

The model is the superposition of the NHPP of 'p' reused and 'q' newly developed components with hard faults and 's' newly developed components with complex faults. Thus, the mean value function of superposed NHPP is

$$m(n)=\sum_{i=1}^{p}m_{ir}(n)+\sum_{j=p+1}^{p+q}m_{jr}(n)+\sum_{k=p+q+1}^{p+q+s}m_{kr}(n)$$

or

$$\begin{aligned}m(n)=&\sum_{i=1}^{p}a_i(1-(1-\delta b_i)^n)+\sum_{j=p+1}^{p+q}a_j\frac{1-(1+\delta b_jn)(1-\delta b_j)^n}{1+\beta(1-b_j)^n}\\&+\sum_{k=p+q+1}^{p+q+s}a_k\frac{1-\left(1+\delta b_kn+\frac{b_k^2n\delta(n+1)\delta}{2}\right)(1-\delta b_k)^n}{1+\beta(1-b_k)^n}\end{aligned} \tag{9.6.10}$$

where $\sum_{i=1}^{p+q+s}a_i=a$ (the total fault content of the software). Note that a distributed system can have any number of used and newly developed components. The equivalent continuous can be derived taking limit $\delta\to 0$, i.e.

$$m(n) \to m(t) = \left(\sum_{i=1}^{p} a_i \left(1 - e^{-bt}\right) + \sum_{j=p+1}^{p+q} a_j \frac{1-(1+bt)e^{-bt}}{1+\beta e^{-bt}} + \sum_{k=p+q+1}^{p+q+s} a_k \frac{1-\left(1+bt+\left(b^2t^2\right)/2\right)e^{-bt}}{1+\beta e^{-bt}} \right)$$

The continuous model is due to [19].

9.7 Discrete Change Point Software Reliability Growth Modeling

Change point analysis for software reliability modeling is of predominant interest. Although changes are observed in almost every process and system but for the software reliability analysis phenomenon of change is a common observation. Software reliability growth during testing phase depends on a number of factors and changes either forced or continuous are observable in many of these factors such as testing strategy, defect density, testing efficiency, testing environment etc. Consideration of change point provides significant improvement in the reliability prediction. Several change point SRGM have been discussed in Chap. 5 considering diverse testing phenomena. Some studies in change point based software reliability modeling also focus on discrete time space. This section gives an insight into how to develop discrete time change point based SRGM.

9.7.1 Discrete S-Shaped Single Change Point SRGM

The delayed s-shaped discrete SRGM discussed in Sect. 9.2.3 due to [7] can be derived alternatively in one stage as follows

$$\frac{m_r(n+1) - m_r(n)}{\delta} = b(n)(a - m_r(n)), \tag{9.7.1}$$

where $b(n) = \frac{b^2(n+1)}{1+bn}$.

Change point modeling in software reliability is based on the parametric variability modeling approach, i.e. to incorporate the phenomena of change, fault detection rate before the change point is assumed to be different from the fault detection rate after change-point. Under the basic assumption, the expected cumulative number of faults removed between the nth and the $(n+1)$th test cases is proportional to the number of faults remaining after the execution of the nth test run, satisfies the following difference equation (Kapur et al. [20])

$$\frac{m(n+1) - m(n)}{\delta} = b(n)(a - m(n)), \tag{9.7.2}$$

where

$$b(n) = \begin{cases} \frac{b_1^2(n+1)}{1+b_1 n} & 0 \le n < \eta_1 \\ \frac{b_2^2(n+1)}{1+b_2 n} & n \ge \eta_1 \end{cases} \tag{9.7.3}$$

and η_1 is the test case number from whose execution onward change in the fault detection rate is observed.

Case 1 $(0 \le n < \eta_1)$

Solving the difference equation (9.7.2) substituting $b(n)$ from (9.7.3), under the initial condition at $n = 0$, $m(n) = 0$, we get

$$m(n) = a(1 - (1 + \delta b_1 n)(1 - \delta b_1)^n) \tag{9.7.4}$$

The equivalent continuous of (9.7.4) can be derived taking limit $\delta \to 0$,

$$m(n) = a(1 - (1 + \delta b_1 n)(1 - \delta b_1)^n) \to a\left(1 - (1 + bt)e^{-bt}\right)$$

Case 2 $(n \ge \eta_1)$

Substituting the fault detection rate applicable after the change point in difference equation (9.7.2) and using the initial condition at $n = \eta_1$, $m(n) = m(\eta_1)$, we get

$$m(n) = a\left[1 - \frac{(1 + \delta b_1 \eta_1)}{(1 + \delta b_2 \eta_1)}(1 + \delta b_2 n)(1 - \delta b_2)^{(n-\eta_1)}(1 - \delta b_1)^{\eta_1}\right] \tag{9.7.5}$$

The equivalent continuous of (9.7.6) can be derived taking limit $\delta \to 0$

$$m(n) \to m(t) = a\left(1 - \left(\frac{1 + b_1 t_1}{1 + b_2 t_1}\right)(1 + b_2 t)e^{-(b_1 t_1 + b_2 (t - t_1))}\right)$$

9.7.2 Discrete Flexible Single Change Point SRGM

The change point model discussed in the previous section produces a pure s-shaped model. We know that flexible models are always preferred for a variety of real life applications as they can describe both exponential as well as s-shaped failure curves. Besides flexible models also well captures the variability of the s-shaped curves. In this section we describe a flexible change point SRGM which can support a wider range of practical applications [8].

Assuming a logistic function with parameter variability defines the fault detection rate before and after the change point the difference equation for the model under the general assumptions of the discrete NHPP models is formulated as

$$\frac{m_r(n+1) - m_r(n)}{\delta} = b(n)(a - m_r(n)) \tag{9.7.6}$$

where

$$b(n) = \begin{cases} \frac{b_1}{1+\beta(1-b_1\delta)^n} & 0 \le n < \eta_1 \\ \frac{b_2}{1+\beta(1-b_2\delta)^n} & n \ge \eta_1 \end{cases} \tag{9.7.7}$$

The mean value function of the SRGM based on the difference equation (9.7.6) and the fault detection rate defined by (9.7.7) under the initial conditions at $n = 0$, $m(n) = 0$; and $n = \eta_1$, $m(n) = m(\eta_1)$ is given by

$$m(n) = \begin{cases} a\left(\frac{1-(1-b_1\delta)^n}{1+\beta(1-b_1\delta)^n}\right) & 0 \le n < \eta_1 \\ a\left(\frac{1-(1+\beta)(1+\beta(1-b_2\delta)^{n_1})(1-b_1\delta)^{n_1}(1-b_2\delta)^{n-n_1}}{(1+\beta(1-b_1\delta)^{n_1})(1+\beta(1-b_2\delta)^n)}\right) & n \ge \eta_1 \end{cases} \tag{9.7.8}$$

For $\beta = 0$ this model produces an exponential failure curve while for other values of $\beta > 0$ of it produces an S-shaped curve. The continuous model equivalent to the flexible change point model can be derived taking limit $\delta \to 0$.

$$m(n) \to m(t) = \begin{cases} a\left(1 - \frac{(1+\beta)e^{-b_1 t}}{1+\beta e^{-b_1 t}}\right) & 0 \le t \le \tau, \\ a\left(1 - \frac{(1+\beta)\left(1+\beta e^{-b_2\tau}\right)}{(1+\beta e^{-b_1\tau})(1+\beta e^{-b_2 t})} e^{-b_1\tau - b_2(t-\tau)}\right) & \tau < t \end{cases}$$

9.7.3 An Integrated Multiple Change Point Discrete SRGM Considering Fault Complexity

All the fault complexity based models discussed in the book up to now have been formulated on the assumption that the nature of the failure and fault removal process for each type of fault remain same throughout the testing process. There are many factors that affect software testing. These factors are unlikely to be kept stable during the entire process of software testing, with the result that the underlying statistics of the failure process is likely to experience changes. The fault detection/removal rates for all the faults lying in the software differ on the basis of their complexity. While modeling for simple faults it is assumed that using a constant fault detection rate the failure and removal process of simple faults can be described in single stage. The assumption of constancy of detection rate may not hold true in many situations. All the factors that bring changes in the overall testing process can also be operating simultaneously or due to some abrupt changes brought forcefully in the testing process can change the testing processes of simple faults. The similar arguments hold true even for other types of faults. Hence it is a wise thinking to describe the failure and removal phenomena of each type of fault on the change point concept.

The model discussed in this section [21] first develops a general model on the basis of above arguments, i.e. it first describes how to model the testing process of

different types of faults on the change point concept for the case of n change points. Then following the general formulation, two special cases are developed for two and three change points, respectively, and models for total testing processes are developed. In this model the removal phenomena of each type of fault in each change point interval are derived in single stage. Firstly we explain the readers how we can develop the mean value functions for the different types of faults directly in one stage considering the delay of the removal process.

The two-stage removal process for the hard faults in (9.5.6) and (9.5.13) can be derived in one stage directly assuming fault detection rate per remaining fault to be

$$\hat{b}_2(n+1) = \frac{b_2^2(n+1)}{(1+b_2(n+1))} \tag{9.7.9}$$

$$\hat{b}_2(n+1) = \frac{b_2(1+\beta_2+b_2(n+1)) - b_2(1+\beta_2(1-b_2)^{n+1})}{(1+\beta_2(1-b_2)^{n+1})(1+\beta_2+b_2 n)} \tag{9.7.10}$$

in the difference equation

$$m_{2r}(n+1) - m_{2r}(n) = \hat{b}_2(n+1)(a_2 - m_{2r}(n)) \tag{9.7.11}$$

Similarly we can define the removal rates for the complex or other types of the faults and derive the time lag models in the single stage. On these lines we formulate general change point based fault complexity SRGM.

The difference equation describing the model can be given as

$$m_i^{'}(n) = b_{ij}[ap_i - m_i(n)] \quad i = 1,\ldots,k;\ j = 1,\ldots,q \tag{9.7.12}$$

where

$$bij = \begin{cases} b_{i1}(n) & 0 \le n \le \eta_1 \\ b_{i2}(n) & \eta_1 < n \le \eta_2 \\ \ldots & \\ \ldots & \\ b_{iq}(n) & n > \eta_q \end{cases} \tag{9.7.13}$$

Index i denotes the type of fault and j corresponds to the change point. k, q is the number of fault types and change points respectively.

The exact solution of the above model equations can be obtained on substituting the functional forms of the fault removal rates in (9.7.13) and defining the number of change points based on past data or by the experience. Now the mean value function of the expected total number of faults removed from the system is given as

$$m(n) = \sum_{i=1}^{r} m_i(n) \tag{9.7.14}$$

Table 9.3 Fault removal rates for the fault complexity based change point SRGM

Time interval/type of fault	Simple Fault detection rates	Hard	Complex
$0 \le n \le \eta_1$	b_{11}	$\frac{b_{21}^2 n}{1+b_{21}n}$	$\frac{\left(b_{31}^3 n(n+1)\right)/2}{1+b_{31}n+\left(b_{31}^2 n(n+1)\right)/2}$
$\eta_1 < n \le \eta_2$	b_{12}	b_{22}	$\frac{b_{32}^2 n}{1+b_{32}n}$
$n > \eta_2$	b_{13}	b_{23}	b_{33}

Various diverse testing environment and testing strategies existing for different types of software can be analyzed from the above model by choosing the appropriate forms of the fault removal rates $b_{ij}(t)$ (based on the past failure data and experience of the developer). One of the most simple and general case would be the one when we consider fault removal rates for each type of fault in each change point interval to be constant but distinct for each i and j, if we observe exponential failure curve growth pattern for each type of fault. However in case of general purpose software we may expect that the fault removal rates for each type of fault may increase with time as the testing team gains experience with the code and learning occurs and reaches a certain constant level toward the end of the testing phase.

The fault detection rate of hard and/or complex faults is slightly less than that of simple fault type. We may also observe a decreasing FRR toward the end of testing phase since most of the faults lying in the software are removed and failure intensity has become very less. Increasing and/or decreasing trend in FRR can be depicted with the time dependent forms of $b_{ij}(t)$. From the study of various fault detection rates used in reliability growth modeling we summarize in Table 9.3 the fault removal rates of a model referring to three types of faults with two change points.

In the above definition of the fault removal rates it is assumed that due to experience and learning process the removal rate increases after the first change point η_1, further it is assumed that after the second change point η_2 the removal process is described by constant rates as the testing skill of the testing personnel reaches a level of stability. Now using the difference equation (9.7.12) and the fault removal rates defined in Table 9.3 we compute the mean value functions for the removal process for each type of fault.

9.7.3.1 Model for Simple Faults

The mean value function for the simple faults using the initial conditions $n = 0$, $m_1(n) = 0$ $n = \eta_1$, $m_1(n) = m_1(\eta_1)$ and $n = \eta_2$, $m_1(n) = m_1(\eta_2)$, respectively, in each change point interval is given as

$$m_1(n) = \begin{cases} a_1(1-(1-b_{11})^n) & 0 \le n \le \eta_1 \\ a_1[1-(1-b_{11})^{\eta_1}(1-b_{12})^{n-\eta_1}] & \eta_1 < n \le \eta_2 \\ a_1[1-(1-b_{11})^{\eta_1}(1-b_{12})^{\eta_2-\eta_1}(1-b_{13})^{n-\eta_2}] & n > \eta_2 \end{cases} \quad (9.7.15)$$

9.7.3.2 Model for Hard and Complex Faults

Using the same initial conditions as in case of simple faults the mean value function for the hard and complex faults in each change point interval is given respectively as

$$m_2(n) = \begin{cases} a_2(1-(1+b_{21}n)(1-b_{21})^n) & 0 \le n \le \eta_1 \\ a_2[1-(1+b_{21}\eta_1)(1-b_{21})^{\eta_1}(1-b_{22})^{n-\eta_1}] & \eta_1 < n \le \eta_2 \\ a_2[1-(1+b_{21}\eta_1)(1-b_{21})^{\eta_1}(1-b_{22})^{\eta_2-\eta_1}(1-b_{23})^{n-\eta_2}] & n > \eta_2 \end{cases} \tag{9.7.16}$$

$$m_3(n) = \begin{cases} a_3\left(1-\left(1+b_{31}n+\left(b_{31}^3 n(n+1)\right)/2\right)(1-b_{31})^n\right) & 0 \le n \le \eta_1 \\ a_3\left[1-\frac{(1+b_{32}n)}{(1+b_{32}\eta_1)}\left(\begin{array}{r} 1+b_{31}\eta_1+ \\ \frac{b_{31}^2\eta_1(\eta_1+1)}{2} \end{array}\right)(1-b_{31})^{\eta_1}(1-b_{32})^{n-\eta_1}\right] & \eta_1 < n \le \eta_2 \\ a_2\left[1-\frac{(1+b_{32}\eta_2)}{(1+b_{32}\eta_1)}\left(\begin{array}{r} 1+b_{31}\eta_2 \\ +\frac{b_{31}^2\eta_1(\eta_1+1)}{2} \end{array}\right)\left(\begin{array}{r} (1-b_{31})^{\eta_1}(1-b_{32})^{\eta_2-\eta_1} \\ (1-b_{33})^{n-\eta_2} \end{array}\right)\right] & n > \eta_2 \end{cases} \tag{9.7.17}$$

The mean value function of the total fault removal formula is formulated using the mean value functions defined in (9.7.15), (9.7.16) and (9.7.17) given as

$$m(n) = m_1(n) + m_2(n) + m_3(n) \tag{9.7.18}$$

The fault removal rates in Table 9.3 depicts a particular case, where initially the removal rate is slow than it accelerate and ultimately reaches a stable value for both hard and complex faults. This is a situation observed commonly due to the learning process of the testing. Various other combinations of fault removal rates may be chosen to describe the particular application in consideration. The current state-of-the-art in discrete software reliability growth modeling is illustrated in this chapter. The development of discrete modeling is still immature. Lot of scope of research is left in the field together with research on the new techniques which can resolve the problem of mathematical complexity in this type of model development.

9.8 Data Analysis and Parameter Estimation

Starting from Chaps. 2–8 we emphasize on the development and application of continuous time software reliability growth models. Majority of study in reliability modeling lies in the continuous time models that form the very reason why practitioner prefer to use continuous time models. Mathematical complexity and complicated functional forms of discrete models are among the other reason. Analysis on discrete modeling suggests that if the software failure data relates

number of test runs to the failure observed then it is more justified to apply a discrete model then a continuous one to obtain more authenticated and accurate result. On the other hand use of statistical software to carry out the model analysis and parameter estimation makes not much difference if one uses a continuous model or a discrete model in terms of mathematical complexity. This section of the chapter is focused on the data analysis and parameter estimation of several selected models discussed here.

Failure Data Set

The failure data is cited in [22]. The data is obtained during testing of software that runs on a wireless network product and the software ran on an element within a wireless network switching centre. The software testing observation period is for 51 weeks during which a total of 203 faults are observed. The software is tested for 1,001 days and a total of 181 failures were observed. The data is recorded in calendar time. While developing a discrete model we defined $t = n\delta$, where, δ is a constant and represents the average time difference interval between consecutive test runs. For the sake of simplicity assume $\delta = 1$ then the calendar time data can be assumed to represent the test run data. So we treat this data as the one where 51 test runs are executed, which resulted in 203 failures during testing phase.

Following discrete models discussed in the chapter have been chosen for data analysis and parameter estimation.

Model 1 (M1) Discrete Exponential Model [4]

$$m(n) = a(1 - (1 - b\delta)^n)$$

Model 2 (M2) Modified Discrete Exponential Model [4]

$$m(n) = m_1(n) + m_2(n) = \sum_{i=1}^{2} a_i(1 - (1 - b_i\delta)^n)$$

Model 3 (M3) Discrete Delayed S–Shaped Model [13]

$$m_r(n) = a[1 - (1 + bn\delta)(1 - b\delta)^n]$$

Model 4 (M4) Discrete SRGM with Logistic Learning Function [23]

$$m(n) = \frac{a}{1 + \beta(1 - b\delta)^n}[1 - (1 - b\delta)^n]$$

Model 5 (M5) Discrete SRGM for Error Removal Phenomenon [9]

$$m_r(n) = a\left[\frac{1 - \{1 - \delta(p + q)\}^n}{1 + (q/p)\{1 - \delta(p + q)\}^n}\right]$$

Model 6 (M6) Discrete SRGM under Imperfect Debugging Environment [12]

$$m_r(n) = \frac{a_0 b_0 p\delta}{1 + \beta(1 - b_0 p\delta)^n}\left[\frac{(1 + \alpha\delta)^n - (1 - b_0 p\delta)^n}{(\alpha\delta + b_0 p\delta)}\right]$$

Model 7 (M7) Discrete S-shaped Single Change Point SRGM Kapur et al. [20]

$$m(n) = \begin{cases} a(1-(1+\delta b_1 n)(1-\delta b_1)^n) & 0 \le n < \eta_1 \\ a\left[1-\frac{(1+\delta b_1 \eta_1)}{(1+\delta b_2 \eta_1)}(1+\delta b_2 n)(1-\delta b_2)^{(n-\eta_1)}(1-\delta b_1)^{\eta_1}\right] & n \ge \eta_1 \end{cases}$$

Model 8 (M8) Discrete Flexible Single Change Point SRGM [8]

$$m(n) = \begin{cases} a\left(\frac{1-(1-b_1\delta)^n}{1+\beta(1-b_1\delta)^n}\right) & 0 \le n < \eta_1 \\ a\left(\frac{1-(1+\beta)(1+\beta(1-b_2\delta)^{n_1})(1-b_1\delta)^{n_1}(1-b_2\delta)^{n-n_1}}{(1+\beta(1-b_1\delta)^{n_1})(1+\beta(1-b_2\delta)^n)}\right) & n \ge \eta_1 \end{cases}$$

The results of data analysis and estimated parameters of the models M1–M8 are listed in Table 9.4. The goodness of fit curve is shown in Fig. 9.1.

The goodness of fit curve clearly reveals the s-shaped nature of failure curve on the test run axis. This observation agrees with the result of data analysis as the mean square error of fitting is very high for both of the exponential models (M1 and M2) as compared to the other s-shaped and flexible models. The fault detection rates for fault-types I and II come out to be same and the total fault content ($a_1 + a_2 = 149 + 152 = 301$) coincides with the exponential model M1, which means that failure process of software does not require the distinction on the basis of fault complexity for its description. MSE and R^2 measures corresponding

Table 9.4 Data analysis results of models M1–M8

Model	Estimated parameters					Comparison criteria	
	a,a_1	a_2,p	b,b_1,	b_2,q, α	β	MSE	R^2
M1	301	–	0.0240	–	–	79.56	0.981
M2	149	152	0.0240	0.0240	–	76.44	0.981
M3	215	–	0.0862	–	–	14.13	0.997
M4	206	–	0.1015	–	4.681	6.67	0.998
M5	206	0.0179	–	0.0837	–	6.67	0.998
M6	206	0.9700	0.1044	0.0000	4.655	6.67	0.998
M7	235	–	0.0790	0.0478	–	10.36	0.997
M8	242	–	0.0771	0.7010	3.393	18.44	0.996

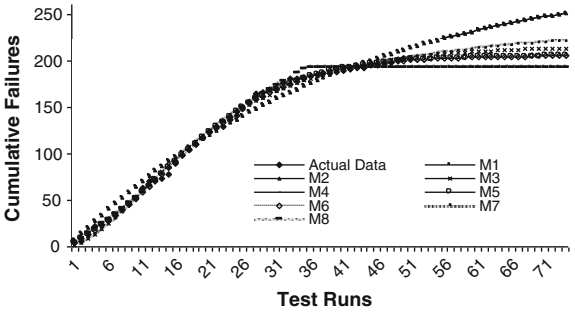


Fig. 9.1 Goodness of fit curves for models M1–M8

to models M4, M5 and M6 overlap each other. This can be interpreted as all these models are flexible models, If we assume $(p + q) = b$ and $(q/p) = \beta$ as in case of continuous counterparts then there remains no difference between the models M4 and M5. But both of the models have different assumptions and interpretations. Although it seems unrealistic but the parameter α in model M6 comes out to be zero denying the presence of error generation, which again makes the results of model M6 equivalent to those of model M4. Hence any of the models M4–M6 can be chosen for further analysis and represent the testing process, the choice can be subjective of the decision maker. One more interesting observation can be made from the data analysis results is that both of the change point models are not among the best fits models illustrating a smooth testing process up to the point of data observation, i.e. 51 test runs.

We have seen the data analysis results reveal the presence of only one type of faults in the data and no change point, hence we choose a different data set to show the application of fault complexity and integrated change point and fault complexity discrete models.

9.8.1 Application of Fault Complexity Based Discrete Models

Failure Data Set

The failure data is also cited in [24]. This data set is obtained during from a real software project on Brazilian Switching system, TROPICO R-1500, for 1,500 telephone subscribers. The software size is about 300 Kb written in assembly language. During the 81 weeks of software testing 461 faults were removed. Again we assume the data is corresponding to the 81 test runs instead of calendar time execution period for the reasons stated above.

Following models discussed in the chapter have been chosen

Model 9 (M9) Generalized Discrete Erlang SRGM [1]

$$m(t) = a_1(1-(1-b_1)^n) + a_2(1-(1+b_2n)(1-b_2)^n) + a_3\left(1-\left(\left(1+b_3n+\left(b_3^2n(n+1)/2\right)\right)\right)(1-b_3)^n\right)$$

Model 10 (M10) Discrete SRGM with faults of Different Complexity Incorporating Logistic Learning Function [15]

$$m(t) = a_1(1-(1-b_1)^n) + a_2\frac{1-(1+b_2n)(1-b_2)^n}{1+\beta(1-b_2)^n} + a_3\frac{1-(1-b_3n+\frac{b_3^2n(n+1)}{2})(1-b_3)^n}{1+\beta(1-b_3)^n}$$

Model 11 (M11) Integrated Multiple Change Point Discrete SRGM Considering Fault Complexity [21]

Models for Simple, Hard and Complex Faults are, respectively, defined as

$$m_1(n) = \begin{cases} a_1(1-(1-b_{11})^n) & 0 \le n \le \eta_1 \\ a_1[1-(1-b_{11})^{\eta_1}(1-b_{12})^{n-\eta_1}] & \eta_1 < n \le \eta_2 \\ a_1[1-(1-b_{11})^{\eta_1}(1-b_{12})^{\eta_2-\eta_1}(1-b_{13})^{n-\eta_2}] & n > \eta_2 \end{cases}$$

$$m_2(n) = \begin{cases} a_2(1-(1+b_{21}n)(1-b_{21})^n) & 0 \le n \le \eta_1 \\ a_2[1-(1+b_{21}\eta_1)(1-b_{21})^{\eta_1}(1-b_{22})^{n-\eta_1}] & \eta_1 < n \le \eta_2 \\ a_2[1-(1+b_{21}\eta_1)(1-b_{21})^{\eta_1}(1-b_{22})^{\eta_2-\eta_1}(1-b_{23})^{n-\eta_2}] & n > \eta_2 \end{cases}$$

$$m_3(n) = \begin{cases} a_3\left(1-\left(1+b_{31}n+\left(b_{31}^3 n(n+1)\right)/2\right)(1-b_{31})^n\right) & 0 \le n \le \eta_1 \\ a_3\left[1-\dfrac{(1+b_{32}n)}{(1+b_{32}\eta_1)}\left(\begin{matrix}1+b_{31}\eta_1+\\ \dfrac{b_{31}^2\eta_1(\eta_1+1)}{2}\end{matrix}\right)(1-b_{31})^{\eta_1}(1-b_{32})^{n-\eta_1}\right] & \eta_1 < n \le \eta_2 \\ a_2\left[1-\dfrac{(1+b_{32}\eta_2)}{(1+b_{32}\eta_1)}\left(\begin{matrix}1+b_{31}\eta_2\\ +\dfrac{b_{31}^2\eta_1(\eta_1+1)}{2}\end{matrix}\right)\left(\begin{matrix}(1-b_{31})^{\eta_1}(1-b_{32})^{\eta_2-\eta_1}\\ (1-b_{33})^{n-\eta_2}\end{matrix}\right)\right] & n > \eta_2 \end{cases}$$

$$m(n) = m_1(n) + m_2(n) + m_3(n)$$

The results of data analysis and estimated parameters of the models M9–M11 are listed in Table 9.5. The goodness of fit curve is shown in Fig. 9.2.

Data analysis results depict that the fault complexity models which incorporate the learning phenomenon (M10) best describe this data set among the chosen models, which show a high value of the learning parameter $\beta = 598.11$. They also depict that the largest population of faults in the software is among the simple category (59.04%) and rest of the faults are of hard and complex category.

Table 9.5 Data analysis results of models M9–M11

Model	Estimated parameters							Comparison criteria	
								MSE	R^2
M9	187	208	205	0.0471	0.0681	0.0272	–	83.39	0.997
	(a_1)	(a_2)	(a_3)	(b_1)	(b_2)	(b_3)			
M10	285	96	102	0.0457	0.2341	0.1025	598.11	41.19	0.998
	(a_1)	(a_2)	(a_3)	(b_1)	(b_2)	(b_3)	(β)		
M11	184	219	129	0.95586	0.00041	0.00029	–	102.67	0.995
	(a_1)	(a_2)	(a_3)	(b_{11})	(b_{21})	(b_{31})			
	0.99959	0.97071	0.00268	0.00041	0.00041	0.02592	–		
	(b_{12})	(b_{22})	(b_{32})	(b_{13})	(b_{23})	(b_{33})			

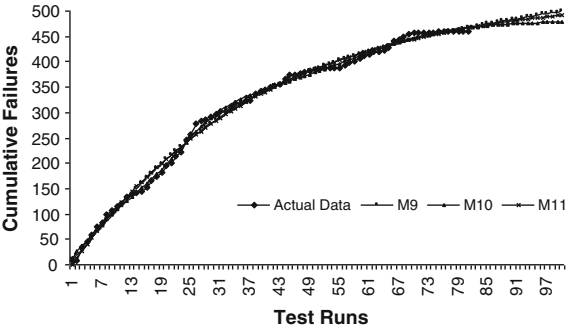


Fig. 9.2 Goodness of fit curves for models M9–M11

Exercises

1. Describe a non-homogeneous Poisson process in discrete time space.
2. What are the merits and demerits of formulating SRGM in discrete time space over the continuous time models?
3. The difference equation of the discrete time exponential SRGM is given by $\frac{m(n+1)-m(n)}{\delta} = b(a - m(n))$. Use the method of PGF to derive the mean value function of the SRGM.
4. Derive the mean value function of single change point exponential model. Use the software failure data of a real testing process of software given below and estimate the unknown parameters of the exponential single change point model and other single change point model discussed in the chapter. Which model fits best to this data set? Assume a change point occurs at the execution time 656 after the detection of 21 faults.

Failure number	Exposure time (hrs)	Failure number	Exposure time (hrs)
1	24	17	520
2	80	18	520
3	96	19	560
4	120	20	632
5	200	21	656
6	200	22	656
7	201	23	776
8	224	24	888
9	256	25	952
10	424	26	952
11	424	27	976
12	456	28	992
13	464	29	1144
14	488	30	1328
15	488	31	1360
16	496		

5. Section 9.7.3 describes general formulation for an integrated multiple change point discrete SRGM considering fault complexity. A two change model is developed to describe the testing process for software containing three types of faults. Suppose during the testing process three change points are observed and the fault detection rates for the three types of faults can be expressed as listed in the following table.

Time interval/ Type of fault	Simple Fault detection rates	Hard	Complex
$0 \le n \le \eta_1$	b_{11}	b_{21}	b_{31}
$\eta_1 < n \le \eta_2$	b_{12}	$\frac{b_{22}^2 n}{1 + b_{22} n}$	$\frac{\left(b_{32}^3 n(n+1)\right)/2}{1 + b_{32} n + \left(b_{32}^2 n(n+1)\right)/2}$
$\eta_2 < n \le \eta_3$	b_{13}	b_{23}	$\frac{b_{33}^2 n}{1 + b_{33} n}$
$n > \eta_3$	b_{14}	b_{24}	b_{34}

Develop an SRGM to describe the above situation.

References

1. Kapur PK, Younes S (1995) Software reliability growth model with error dependency. Microelectron Reliab 35(2):273–278
2. Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware and software reliability. World Scientific, Singapore
3. Kapur PK, Jha PC, Singh VB (2008) On the development of discrete software reliability growth models. In: Misra KB (ed) Handbook on performability engineering. Springer, 1239–1254.
4. Yamada S, Osaki S (1985) Discrete software reliability growth models. Appl Stoch Models Data Anal 1:65–77
5. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Tran Reliab R 28(3):206–211
6. Kapur PK, Bai M, Bhushan S (1992) Some stochastic models in software reliability based on NHPP. In: Venugopal N (ed) Contributions to stochastics. Wiley Eastern Limited, New Delhi.
7. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Tran Reliab R 32(5):475–484
8. Kapur PK, Goswami DN, Khatri SK, Johri P (2007c) A flexible discrete software reliability growth model with change-point. In: Proceedings of the national conference on computing for nation development, INDIACom- 2007, pp 285–290
9. Kapur PK, Gupta A, Gupta A, Kumar A (2005) Discrete software reliability growth modeling. In: Kapur PK, Verma AK (eds) Quality, Reliability and IT (Trends & Future Directions). Narora Publications Pvt. Ltd., New Delhi, pp 158–166
10. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
11. Bardhan AK (2002) Modeling in software reliability and its interdisciplinary nature. Ph.D. Thesis, University of Delhi, Delhi

12. Kapur PK, Singh OP, Shatnawi O, Gupta A (2006e) A discrete NHPP model for software reliability growth with imperfect fault debugging and fault generation. Int. J Performability Eng 2(4):351–368
13. Kapur PK, Agarwal S, Garg RB (1994) Bi-criterion release policy for exponential software reliability growth models. Recherche Operationanelle/Oper Res 28:165–180
14. Kapur PK, Younes S, Agarwala S (1995) Generalized Erlang software reliability growth model. ASOR Bull 35(2):273–278
15. Kapur PK, Shatnawi O, Singh O (2005) Discrete time fault classification model. In: Kapur PK, Verma AK (eds) Quality, reliability and IT (Trends & Future Directions). Narora Publications Pvt. Ltd., New Delhi, pp 132–145
16. Shatnawi O, Kapur PK (2008) A generalized software fault classification. WSEAS Tran Comput 7(9):1375–1384
17. Kapur PK, Gupta A, Singh OP (2005b) On discrete software reliability growth model and categorization of faults. OSEARCH 42(4):340–354
18. Kapur PK, Singh OP, Kumar A, Yamada S (2006d) Discrete software reliability growth models for distributed systems. In: Kapur PK, Verma AK (eds) Proceedings of international conference on quality, reliability and infocom technology. MacMillan Advanced Research Series, pp 101–115
19. Kapur PK, Gupta A, Kumar A, Yamada S (2005) Flexible software reliability growth models for distributed systems. OPSEARCH, J Oper Res Soc India 42(4):378–398
20. Kapur PK, Khatri SK, Jha PC, Johri P (2007) Using change-point concept in software reliability growth. Quality, reliability and infocom technology (Proceedings of ICQRT-2006), Macmillan, India pp 219–230
21. Goswami DN, Khatri SK, Kapur R (2007) Discrete software reliability growth modeling for errors of different severity incorporating change-point concept. Int J Autom Comput 4(4):395–405
22. Jeske DR, Zhang X, Pham L (2005) Adjusting software failure rates that are estimated from test data. IEEE Tran Reliab 54(1):107–114
23. Kapur PK, Anand S, Yadavalli VSS, Beichelt F (2007) A generalised software growth model using stochastic differential equation. Communication dependability quality management. Belgrade, Serbia, pp 82–96
24. Kanoun K, Martini M, Souza J (1991) A method for software reliability analysis and prediction application to the TROPICO-R switching system. IEEE Tran Softw Eng 17(4):334–344

Chapter 10
Software Release Time Decision Problems

Notation

$a(a_i)$	Expected number of faults existing in the software (of fault type i) before testing
$b(b_i)$	Constant fault detection/removal rate per remaining fault per unit time (of fault type i)
$m(t)$ $(m_i(t))$	Expected mean number of failures/removal (of fault type i) by time t, $m(0) = 0$
$m_f(t)$	Expected mean number of failures detected by time t, $m_f(0) = 0$
$m_r(t)$	Expected mean number of faults removed by time t, $m_r(0) = 0$
p	Probability of perfect debugging of a fault, $0 < p < 1$
α	Constant rate of error generation, $0 < \alpha < 1$
$\lambda(t)$	Failure intensity function $\lambda(t) = m'(t)$
$C_1(C_1^{'})$	Cost incurred on perfect (imperfect) debugging of fault during testing phase
$C_2\left(C_2'\right)$	Cost incurred on a perfect (imperfect) debugging of fault after release of the software system $\left(C_2 > C_1, C_2' > C_1'\right)$
C_3	Testing cost per unit testing time
C_B	Total budget allocated for the software testing
T	Release time of the software
T^*	Optimal release time
R_0	Desired level of software reliability to be achieved at the release time $(0 < R_0 < 1)$
λ_0	Desired level of failure intensity to be achieved at the release time $(0 < \lambda_0 < 1)$
x	The mission time of reliability
$R(x\|T)$	The reliability function
$E(T)$	Expected cost of software systems at time T
Y	Variable representing time to remove an error during testing phase

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_10,
© Springer-Verlag London Limited 2011

μ_y	Expected time to remove an error during testing phase
W	Variable representing time to remove an error during warranty period in operation phase
μ_w	Expected time to remove an error during warranty period in operation phase
T_w	Period of warranty time
T_l	Life cycle of the software
r_i	Proportion of the fault type i in the software, $i = 1, 2$

10.1 Introduction

Reliability, scheduled delivery and cost are the three main quality attributes for almost all software. The primary objective of the software developer's to attain them at their best values, then only they can obtain long-term profits and make a brand image in the market for longer survival. The importance of reliability objective has escalated many folds as it is a user-oriented measure of quality. Other reasons being, diversified implementation of software in the various domains around the world, critical dependency of the various systems worldwide on computing systems, global trades, highest order growth in the information technology and competition. Notwithstanding its unassailable value, there is still no way to test whether software is completely fault-free or can be made fault-free so that the highest possible value of reliability can be attained how long the testing is continued. On the other hand software users' requirements conflict with the developers. Software users demand faster deliveries, cheaper software and quality product, whereas software developers aim at minimizing their development cost, maximizing the profit margins and meeting the competitive requirements. The resulting situation calls for tradeoffs between conflicting objectives of software users' requirements with the developers. As a course of best alternative the developer management must determine optimally when to stop testing and release the software system to the user focusing on the users' requirements, simultaneously satisfying their own objectives. Such a problem is known as software release time decision (SRTD) problem in the literature of software reliability engineering. Timely release of software provides dual advantage to the developers. First, they obtain maximum returns on their investments, reduce the development costs, meet the competitive goals and increase the organizational goodwill in the market. Second, they can satisfy the conflicting user requirements if the software release time is determined by minimizing the total software cost whereas the goal of reliability is achieved, etc. This implies the advantage of offering the software at an economic price with higher quality level. Delay in software release imposes the burden of penalty cost\revenue loss and the product may suffer from obsolescence in the market. In contrast to this in case of a premature release the developer may have to spend lot of time and effort to fix the software faults after release and suffer

from goodwill loss. Hence one must determine the optimal time to release the software before launching in order to reduce the dual losses that can be imposed on the developers related to both early release and late release. Such a problem of software reliability engineering discipline can be formulated as an optimization problem with single or multiple objectives under some well-defined sets of system, technical and management or user-defined constraints.

Operational research has its primary concern with the formulation of mathematical models. A *mathematical model* is an abstraction of the assumed real world, expresses in an amenable manner the mathematical functions that represent the behavior of the assumed system [1]. A model can be developed with respect to a system to measure some particular quantity of interest such as a cost model or a profit model or it may represent the assumed system as a whole used to optimize the system performance, the *optimization model*. The optimization models developed for the engineering and business professional allow them to choose the best course of action and experiment with the various possible alternative decisions. The software reliability growth models developed to estimate and predict software reliability can be used to formulate an optimization model for software release time decision. The field of operational research offers a number of crisp and soft computing methodologies, optimization techniques and routines to solve such problems. Various researchers in the field of software reliability engineering and operational research have formulated different types of release problems and used several optimization techniques depending on the model formulation and application under consideration. This chapter focuses on the formulation of different classes of release time problems, analysis of the formulated problems, problem solution using different optimization techniques both under crisp and soft environment and real life applications of the problems.

The optimal release time is a function of several factors, viz., size, level of reliability desired, skill and efficiency of testing personal, market environment, penalty and/or opportunity loss costs due to delay in release and penalties/warranty cost due to failure in user phase, etc. Software release time determination has remained a prime field of study for many eminent researchers in the field of software engineering, reliability modeling and optimization over the years. Many problems have been formulated and solved by many researchers in the literature [2–18]. The optimization problem of determining the optimal time of software release is mainly formulated based on goals set by the management in terms of cost, reliability and failure intensity, etc. subject to the constraints. It may be noted here that the release time optimization models make use of software reliability growth models to determine the relationship between the testing progress (in terms of cost incurred, failure exposure or reliability growth) and time. Okumoto and Goel [2] derived simplest release time policies based on exponential SRGM in two ways. In the first approach they considered an unconstrained cost objective while in the other, they considered the unconstrained reliability objective. The problem was formulated assuming all the costs (testing and fault removal during testing and operational phases) are deterministic and well defined, as well as the level of reliability required to achieve is determined on the basis of

experience by the management. Later other researchers followed the approach with different considerations and improvements. Most of the problems on release time even up to the recent times have been formulated assuming static conditions. More specifically most of the problems were formulated assuming that the criterion, activity constant coefficients and resource, requirement and structural conditional constants can be computed exactly; the inequalities in the constraints are well defined and remain the same throughout. Crisp optimization techniques such as the method of calculus, Lagrange multipliers or crisp mathematical programming techniques were used to solve the problem. There is a vast literature of crisp SRTD problems. The first part of this chapter focuses on this part of literature and invocates the knowledge of this literature in detail to the readers. The chapter is continued with the discussion of SRTD problems in the fuzzy environment. It is only recently that Kapur et al. [19–20], Jha et al. [21] and Gupta [22] realized the need for formulating the release time problems in the fuzzy environment and also gave many arguments for this reconsideration. In the next paragraph we discuss in detail why a SRTD problem be defined in the fuzzy environment and what procedures are followed to solve such problems.

In the actual software development environment, the computation of various constants of such optimization problem is based on large input data, information processing and past experience. Most of the SRTD problems formulated considering cost, reliability or failure intensity and number of faults removed require the exact computation of cost function coefficients, amount of available resources, reliability/failure intensity aspiration levels, etc. The values of these quantities besides some static factors depend on a number of factors, which are non-deterministic in nature [23]. For example, if we consider components of cost function, i.e. cost per unit testing time and cost of debugging during testing and operational phases, values of their constant coefficients are determined on the basis of setup cost, CPU time and personnel cost. These costs depend on a number of non-static factors such as testing strategy, testing environment, team constitution, skill and efficiency of testing personal, infrastructure, etc. Besides this the software industry suffers from the dilemma that it sees most frequent changes in the team constitutions due to its employees changing jobs frequently. With this, most of the information and data available to the decision makers are ambiguously defined. Due to these stated reasons an exact definition of these costs is not feasible for the developers. Similarly due to conditions prevailing in the market and competitive reasons, the developers can only make ambiguous statements on the organization goals and available resources bringing uncertainty in the problem definition. There are various other sources that bring uncertainty in the computation such as system complexity, subject's awareness, communication and thinking about uncertainty, intended flexibility, complex relationships between the various variables and economics of information, etc. [24]. Actually it is difficult to define the goals and constraints of such optimization problems precisely for practical purposes. One widely accepted solution of this problem is to define the problem under fuzzy environment, as it offers the opportunity to model subjective imagination of the

decision maker and computations of the model constants as precisely as a decision maker and available information is able to describe.

Traditionally the ambiguous information is usually processed to obtain a representative value of the quantity desired assuming a deterministic environment or determined stochastically based on the distribution of sample information due to the absence or lack of actual data or correct method to quantify these techniques. The system goals and constraints are roughly defined, the problem is solved and the solution is implemented on the system [23]. The optimal solution of the problem so obtained is not actually representative of the complete and exact system information. Implementation of such solution may result in huge losses due to a vague definition of the system model. It may be possible that a small violation of a stated goal, given constraint or the model constants may lead to a more efficient and practical solution of the problem [25]. For example, a high level of system reliability is desired to be achieved by the release time, first the definition of the high level is vague, and giving a precise value of target reliability is very difficult due to the randomness present in the progress of testing. Second a small violation in the desired level of reliability may give more efficient solution for the model. Fuzzy set theory [26–28] builds a model to represent a subjective computation of possible effect of the given values on the problem and permits the incorporation of vagueness in the conventional set theory that can be used to deal with uncertainty quantitatively. Fuzzy optimization is a flexible approach that permits a more adequate solution of real problems in the presence of vague information, providing the well-defined mechanisms to quantify the uncertainties directly. It has proven to be an efficient tool for the treatment of fuzzy problems. Another advantage of fuzzy set theory is that it saves lot of time required for enormous information processing in order to determine average values in the classical modeling due to its capability to directly operate on vague information [23].

Fuzzy set theory is a constituent of so-called *soft computing*. The guiding principle of soft computing [29] is to exploit the tolerance for imprecision, uncertainty, partial truth and approximation to achieve tractability; robustness and low solution cost and solve the fundamental problem associated with the current technological development. The principal constituents of soft computing are: fuzzy systems, evolutionary computation, neural networks and probabilistic reasoning. What is important to note is that soft computing is not a mélange. Rather, it is a partnership in which each of the partners contributes a distinct methodology for addressing problems in its domain. Fuzzy theory (fuzzy set and fuzzy logic) plays a leading role in soft computing and this stems from the fact that human reasoning is not crisp. From the times when fuzzy logic and fuzzy set theory were first propounded by Zadeh [26] they emerged as a new paradigm in which linguistic uncertainty could be modeled systematically. The fuzzy set-based optimization was introduced by Bellman and Zadeh [30] in their seminal paper on decision making in a fuzzy environment, in which the concepts of fuzzy constraint, fuzzy objective and fuzzy decision were introduced. These concepts were subsequently used and applied by many researchers. Literature on this exciting field grew by leaps and bounds. Among other fields, optimization was one of the main

beneficiaries of this "revolution". In the last two decades, the principles of fuzzy optimization were critically studied, and the technologies and solution procedures have been investigated within the scope off fuzzy sets. A number of researchers have contributed to the development of fuzzy optimization technique [23, 25, 27, 30–32], etc. Today, similar to the developments in crisp optimization, different kinds of mathematical models have been proposed and many practical applications have been implemented by using the fuzzy set theory in various engineering fields, such as mechanical design and manufacturing [25, 33–34], power systems [35], water resources research [36], control systems [37–38], etc. some of the preliminary concepts of fuzzy set theory are given in Appendix B.

The existing literature of SRTD problem up to recent times is based on the classical optimization methods formulated under crisp environment. On the other hand there are only a few formulations of SRTD problem defined under the fuzzy environment. We know that an SRTD problem can be formulated in a crisp environment or a fuzzy environment. There are no specific guidelines to tell in what situation, which of the two formulations should be adopted. As far as our discussion from the previous sections can be concluded, we may have a notion that fuzzy approach can be preferred in most situations. But in fact this decision depends on a number of factors such as the kind of software under consideration, the nature of data, what information can be made available, in what phase of software development we are in, when looking for this decision, management choice and experience, etc. All, some or many other factors we can think of, can take part in this decision. Research on fuzzy optimization in this field is very limited. This may also contribute to be one of the factors for one to apply the crisp techniques. Well in general there are no hard rules and any of the two can be adopted keeping in mind the suitability of the technique for the project.

10.2 Crisp Optimization in Software Release Time Decision

Considerable amount of work has been done in the literature on the crisp optimization of software release time. Different policies were formulated based on both exponential and s-shaped SRGM in considering different aspects of the software release time. This section in the chapter focuses on a number of problems that have been formulated in the literature and discusses the solution methodologies with numerical examples.

10.2.1 First Round Studies in SRTD Problem

Since the earlier work, the software release time optimization is mainly concerned with cost minimization or reliability maximization. Okumoto and Goel [2] were

the first to discuss the optimal release policy in the literature. They have discussed unconstrained problem with either cost minimization or reliability maximization objective. Yamada and Osaki [3] discussed release time problems with cost minimization objective under reliability aspiration constraint and reliability maximization objective under cost constraint based on exponential, modified exponential and s-shaped SRGM. The following work was mainly concerned with modifying the cost function based on many criteria. Even the simplest cost function for computing the cost of testing and debugging was formulated based on many arguments.

The software performance during the field is dependent on the reliability level achieved during testing. In general, it is observed the longer the testing phase, the better the performance. Better system performance also ensures less number of faults required to be fixed during operational phase. On the other hand prolonged software testing unduly delays the software release as during the later phases of testing, detection and removal of an additional fault results in exponential increase in time and cost suggesting an early release. Considering the two conflicting objectives of better performance with longer testing and reduced costs with early release, Okumoto and Goel [2] formulated the cost function for the total expected cost incurred during testing and debugging in testing and operational phases given as

$$C(T) = C_1 m(T) + C_2(m(T_l) - m(T)) + C_3 T \tag{10.2.1}$$

The cost model (10.2.1) was formulated by Okumoto and Goel with respect to the Goel and Okumoto [39] (refer to Sect. 2.3.1) exponential SRGM. The expected cost function included simple costs such as the cost of isolating and removing an observed fault during testing and operational phases and the cost of testing per unit testing time. They focused on determining the release time by minimizing this cost function. Thus the problem that has been first considered was simply unconstrained minimization of expected cost function.

$$\text{Minimize} \quad C(T) = C_1 m(T) + C_2(m(T_l) - m(T)) + C_3 T \tag{P1}$$

The release time is obtained by differentiating the cost function with respect to time, T and computing the time point where the first derivative is zero based on the method of calculus

$$C'(T) = -(C_2 - C_1)m'(T) + C_3 = 0 \tag{10.2.2}$$

$$\Rightarrow \quad m'(T) = \frac{C_3}{(C_2 - C_1)} \quad \text{where } m'(T) = ab\mathrm{e}^{-bT} \tag{10.2.3}$$

It can be seen that $m'(T) = \lambda(T)$ is a decreasing function in T with $\lambda(0) = ab$ and $\lambda(\infty) = 0$. Based on (10.2.3) the release time can be obtained based on Theorem 10.1.

Theorem 10.1

1. *If* $ab > \frac{C_3}{C_2 - C_1}$ *then* $C'(T) < 0$ *for* $0 < T < T_0$ *and* $C'(T) > 0$ *for* $T > T_0$. *Thus, there exist a finite and unique* $T = T_0 (>0)$ *minimizing the total expected cost.*
2. *If* $ab \leq \frac{C_3}{C_2 - C_1}$ *then* $C'(T) > 0$ *for* $T > 0$ *and hence* $C(T)$ *is minimum for* $T = 0$.

Determining release time with only cost minimization objective becomes purely a developer-oriented policy for software release. Such a decision may not truly prove to be optimization of release time. Release time decision is related to the marketing activities of the software development. In the era of customer-oriented marketing, deciding release time by minimizing the cost of testing and debugging incurred during testing and operational phases may completely ignore the customer requirement of developing software with high reliability. In view of this, the policy of reliability maximization [2] at the release time can give a reasonably affirmative solution. Such a policy for any of the NHPP-based SRGM can be formulated as

$$\text{Maximize} \quad R(x|T) = \exp^{-(m(T+x)-m(T))} \tag{P2}$$

The policy (P2) may require to test the software for an infinite time as reliability is defined as the probability that a software failure does not occur in time interval $(T,\ T + x)$, given that the last failure occurrence time $T \geq 0$ $(x \geq 0)$ is an increasing function of time. But this is not the solution we are looking for as software cannot be tested for infinite time. After a certain time of testing the time required to detect an additional fault increases exponentially which in turn also increases the cost of testing. Consider the case of any firm; no one neither possesses unlimited amount of resources to dispose on testing nor can they continue testing for infinite time. For such a policy we can specify a target level of reliability and release our software at the time point where that level is achieved, irrespective of the cost incurred.

It can be observed that $R(x|0) = \mathrm{e}^{-m(x)}, R(x|\infty) = 1, R(x|T)$ is an increasing function of time, for $T > 0$. Differentiating $R(x|T)$ with respect to T, we have

$$R'(x|T) = \exp^{-(m(T+x)-m(T))}\left(ab\mathrm{e}^{-bT}\left(1 - \mathrm{e}^{-bx}\right)\right) \tag{10.2.4}$$

Since $R'(x|T) > 0 \quad \forall\, T \geq 0$, $R(x|T)$ is increasing for all $T > 0$. Thus, if $R(x|0) < R_0$ there exists $T = T_1 (>0)$ such that $R(x|T_1) = R_0$. Hence the optimal release time policy based on achieving a desired level of reliability R_0 can be determined based on Theorem 10.2.

Theorem 10.2 *Assuming* $T_l > T$

1. *If* $R(x|0) < R_0$ *then* $T^* \geq T_1$, *but* $< T_l$
2. *If* $R(x|0) \geq R_0$ *then* $T^* \geq 0$, *but* $< T_l$.

Both of the former policies considered only one of the aspects of release time; considering any one of them ignores the other. Reliability being a key measure of

quality should be considered keeping in mind the customer's requirement; on the other hand, resources are always limited so that they must be spent judiciously. It is important to have a tradeoff between software cost and reliability. Yamada and Osaki [3] formulated constrained release time problems which minimize the expected software development cost subject to reliability not less than a predefined reliability level or maximize reliability subject to cost not exceeding a predefined budget.

$$\begin{aligned} &\text{Minimize} \quad C(T) \\ &\text{Subject to} \quad R(x|T) \geq R_0 \end{aligned} \tag{P3}$$

Or

$$\begin{aligned} &\text{Maximize} \quad R(x|T) \\ &\text{Subject to} \quad C(T) \leq C_B \end{aligned} \tag{P4}$$

The optimal release time for Problem (P3) and (P4) can be obtained combining the results of Theorems 10.1 and 10.2 for the exponential SRGM according to the Theorems 10.3 and 10.4, respectively.

Theorem 10.3 *Assuming $T_l > T_0$ and $T_l > T_1$ then release time is determined based on following observations, where T_0, T_1 are as defined in theorem (10.1) and (10.2)*

1. *If* $ab > \frac{C_3}{C_2 - C_1}$ *and* $R(x|0) \geq R_0$, *then* $T^* = T_0$.
2. *If* $ab > \frac{C_3}{C_2 - C_1}$ *and* $R(x|0) < R_0$, *then* $T^* = \max(T_0, T_1)$.
3. *If* $ab \leq \frac{C_3}{C_2 - C_1}$ *and* $R(x|0) \geq R_0$, *then* $T^* = 0$.
4. *If* $ab \leq \frac{C_3}{C_2 - C_1}$ *and* $R(x|0) < R_0$ *then* $T^* = T_1$.

Theorem 10.4 *Assume* $T_l > T_0, T_l > T_1, T_l > T_A$ *and* $T_l > T_B$

1. *If* $ab \leq \frac{C_3}{C_2 - C_1}$ *and* $C(0) > C_B$, *or*
2. *If* $ab \leq \frac{C_3}{C_2 - C_1}$ *and* $C(0) < C_B$, *then the budget constraint is met for all* $T(0 \leq T \leq T_A)$, *where* $C(T)_{T=T_A} = C_B$, *then* $T^* = T_A$.
3. *If* $ab > \frac{C_3}{C_2 - C_1}$ *and* $C(T_0) > C_B$, *then more budget is required in order to release the software to meet the above objective*.
4. *If* $ab > \frac{C_3}{C_2 - C_1}$ *and* $C(T_0) < C_B$, *then the budget constraint is met for all* $T(0 \leq T \leq T_A)$, *where* $C(T)_{T=T_B(\geq 0 < T_0)} = C_B$ *and* $C(T)_{T=T_A(>T_0)} = C_B$, *then* $T^* = T_A$.

If obtaining more resources is not a problem or very high level of reliability is required then one can follow (P4) otherwise if the level of reliability required can be fixed prior to this decision, then one can follow (P3). There is no general rule and based on the type of project, one of the above two policies may be used. The Theorems (10.1–10.4) all have been formulated with respect to the exponential

SRGM Goel and Okumoto [39]. Similar studies have been carried out by the authors for the modified exponential as well as s-shaped SRGM. Reader can refer to the original manuscript for details.

Application 10.1

For computing the release time of any software project using any of the above policies first of all the practitioners require the software failure data. Using the collected data we first determine the unknown parameters of the SRGM taken into consideration. Now after obtaining the parameters of the cost and/or reliability function and bounds on the budget or reliability based on the above theorems, we can determine the release time. In this section we describe in detail how a release policy is applied on any collected data. For numerical illustration let us consider the data set from one of the major releases of Tandem Computers software projects [40]; this software is tested for 20 weeks, spending 10,000 CPU hours and 100 faults were observed in this period.

Using this data set the parameters of Goel and Okumoto [39] SRGM are estimated to be $a = 130.30$ and $b = 0.083$. Let us assume the cost parameters to be $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$500$, where C_1 is the cost of removing a fault in the testing phase, C_2 is the corresponding cost for the operational phase and is usually much higher than the cost incurred during testing phase for fault removal, C_3 is the testing cost per unit time and the life cycle of the software is 3 years (156 weeks). It may be noted that removal of software faults in the operational phase is not same as that of testing phase. Fault removal in testing phase is a continuous process while it is not so for operational phase. A number of overheads are incurred before any fault removal in the operational phase. On the other hand the cost of testing is in general very high as compared to fault removal cost. Its components include cost of CPU hours, manpower cost, etc. The software data we are concerned here with are for 20 weeks; it means that the software has already been tested for almost 5 months. Using the estimates of the SRGM we make estimate of the software failures that have been taken place in 20 weeks. The estimation results yield that 105.53 faults have been removed in 20 weeks out of the total number 130.30, thus a total of 24.77 faults are remaining. If testing is continued only for 20 weeks we will spend \$12294.02 in testing and debugging during testing and operational phases and if we assume that the mission time, i.e. $x = 1$, then the software reliability achieved by this time point is 0.1390.

Policy P1

Let us first consider the policy (P1), i.e. cost minimization. In this period of testing using the estimated values of parameters of the SRGM and cost information, we found that $ab \leq C_3/(C_2 - C_1)$. This is the case 2 of Theorem 10.1 which implies that the optimal release time is $T^* = 0$ weeks as the cost function is always increasing (see Fig. 10.1) and minima occur at $T = 0$ weeks. The software possesses the reliability level zero at this release time. Now assume that if the cost of testing, i.e. $C_3 = \$100$; in this case we have obtained a first decreasing and then increasing cost function (Fig. 10.2). The result of case 1 of Theorem 10.1 becomes

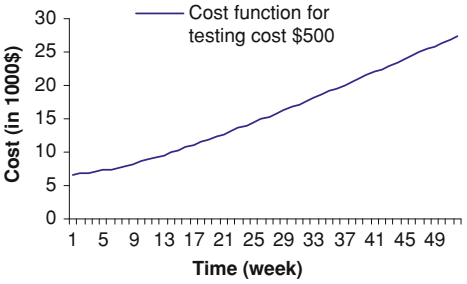


Fig. 10.1 Cost function when testing cost is $500

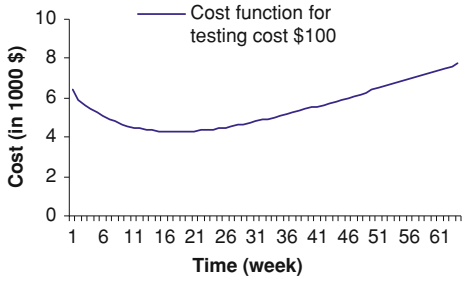


Fig. 10.2 Cost function when testing cost is $100

true, i.e. $ab > C_3/(C_2 - C_1)$ the release time coincides with the point where cost function attains its minimum, i.e. $T_0 = T^* = 17.65$ weeks. At this release time the total cost incurred is $C^* = 4272.46$ and the reliability level achieved is 0.0909. We can see that the results obtained in both of the cases cannot be accepted by the users or developers as when the software would have been released very low level of reliability will be achieved. Such a software would not be suitable for functioning in the operational phase. Let us now compare our results with the policy (P2).

Policy P2

Policy (P1) yields us very low level of reliability. Now consider that the developer wants that the software should not be released in the market prior to the time when the level of reliability is greater than or equal to 0.80, i.e. $R_0 = 0.80$. In this case testing has to be continued at least up to a time period of 46.27 weeks, i.e. software can be released at any time after this period of testing, hence $T^* = 46.27$ weeks. If we consider $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$100$ then \$6035.64 budget is required for achieving the reliability level of 0.80. In Table 10.1 we summarize the results of this policy for the various cases. From the table we can see that for the case when testing cost is \$100 and $R_0 = 0.80$ then the budget consumed is \$6035.64 while if this cost is increased to \$500 then the budget increases to \$24547.53. Hence there is no check on the budget when only reliability aspiration is kept as an objective for the release policy. Results from both of the policies support our findings that unconstrained optimization of either cost minimization or reliability maximization is not sufficient to determine optimally the release time. Figure 10.3 shows the reliability growth curve for the given data.

Table 10.1 Release policies for the reliability maximization policies

R_0	Cost parameters (in $)	T^* (weeks)	Budget consumed (in $)
0.80	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$100$	46.27	6035.64
0.80	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$500$	46.26	24547.53
0.85	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$100$	50.09	6393.56
0.85	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$500$	50.09	26429.56

Policy P3

In the previous sections we have derived the optimal results for both of the unconstrained policies of either cost minimization or reliability maximization. In both of the cases the optimal results obtained may not be acceptable either from the developers' view or the users' view. In case of cost minimization we may end up with very less level of reliability. On the other hand if we keep reliability maximization as our objective then we have no check on the cost and the developers may not have sufficient resources for continuing the testing for long. Now consider that we change our release policy as in problem (P3). Let the developer specify that at least reliability aspiration level be $R_0 = 0.80$ with the cost parameters $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$500$ then in this case we have $ab \leq C_3/(C_2 - C_1)$. Following Theorem 3 we get optimal release time $T^* = 46.26$ weeks. In this time period the cost incurred is $C(T^*) = \$24515.64$. On the other hand if $R_0 = 0.85$ then $T^* = 50.08$ weeks and $C(T^*) = \$26424.63$. If we further change the testing cost C_3 to $C_3 = \$100$ then $ab > C_3/(C_2 - C_1)$. In this case with $R_0 = 0.80$, $T^* = 46.26$ weeks and $C(T^*) = \$6041.08$ and if $R_0 = 0.85$, $T^* = 50.08$ weeks and $C(T^*) = \$6392.63$.

Policy P4

If the developer is not flexible with the budget and wants to achieve maximum level of reliability in the limited amount of resources then he should apply policy (P4) to determine the optimal release policy. If we consider the cost parameters as $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$500$ with a and budget of \$80,000 then we have $T^* = 39.245$ weeks and achieved reliability $R^* = 0.6707$. Table 10.2 summarizes the results of policy (P4) for various cost parameters and budgets.

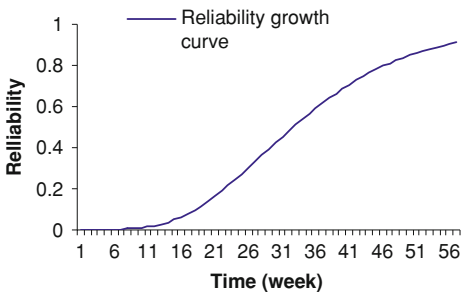


Fig. 10.3 Reliability growth curve for Application 10.1

Table 10.2 Alternative results for policy P4

Budget (in $)	Cost parameters (in $)	T^* (Weeks)	Reliability achieved R^*
80,000	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$500$	39.245	0.6707
6,000	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$100$	45.811	0.7939
1,00,000	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$500$	49.305	0.8409
8,000	$C_1 = \$10$, $C_2 = \$50$ $C_3 = \$100$	66.765	0.9601

Most of the release policies discussed in the literature fall in the category of any one of the problems (P1–P4), i.e. either constrained or unconstrained minimization of cost or maximization of reliability remained primary concern in the release time optimization problem. Some problems also considered maximization of gain or minimization of failure intensity. The cost model of policy (P1) is formulated on simple assumptions. Several modifications have been carried out in the literature in this cost functions to include the penalty or opportunity loss cost due to delivering the software after scheduled time [4, 13], risk cost of failure in field [14], considering random product life cycle [10], expected time of fixing a fault [14], etc. In the next sections of this chapter we will discuss various other release policies formulated on different SRGM and the modifications carried out in the cost function.

10.2.2 A Cost Model with Penalty Cost

In the previous chapters we have discussed that software is either a project type or a product type. Project type software is designed for specific users. Most of the users specify a scheduled delivery time for the delivery of software, making agreement with the developer that if the delivery is delayed then the developer has to pay the penalty cost. Kapur and Garg [4] introduced the concept of releasing the software at scheduled delivery time set by the management and\or with an agreement between the user and developer on release time problem. An expected penalty cost $p_c(t)$ in $(0,T]$ due to delay in the scheduled delivery time is included in the simple cost function (10.2.1) in addition to all the traditional costs. The modified cost function is given as

$$C(T) = C_1 m(T) + C_2(m(T_l) - m(T)) + C_3 T + \int_0^T p_c(T-t)\mathrm{d}G(t) \qquad (10.2.5)$$

The fourth term in the cost model (10.2.5) describes the expected penalty cost in $[T_s, T]$. T_s is the scheduled delivery time assumed to be a random variable with cdf $G(t)$ and finite pdf $g(t)$. Optimal release policies minimizing expected cost subject to the reliability requirement are hence stated as

$$\begin{aligned} \text{Minimize} \quad & C(T) = C_1 m(T) + C_2(m(T_l) - m(T)) + C_3 T \\ & + \int_0^T p_c(T-t)\mathrm{d}G(t) \qquad \text{(P5)} \\ \text{Subject to} \quad & R(x|T) \geq R_0 \end{aligned}$$

The release policy is described for an exponential [39], modified exponential [41] and s-shaped SRGM [42] (refer to Sect. 2.2). Release policies in the previous section have been discussed on the exponential SRGM. In this section we will discuss the release policy with respect to the s-shaped SRGM. For other policies the reader can refer to Kapur and Garg [4].

Differentiating the cost function $C(T)$ with respect to T and equating it to zero, we obtain

$$(C_2 - C_1)\lambda_{m(T)} - \int_0^T \frac{\mathrm{d}p_c(T-t)}{\mathrm{d}T}\mathrm{d}G(t) = C_3 \qquad (10.2.6)$$

where $\lambda_{m(T)} = m'(T) = ab^2 T\mathrm{e}^{-bT}$. It is noted that $\lambda_{m(T)}$ is increasing in $0 \leq T < 1/b$ and decreasing in $1/b < T \leq \infty$.

Case (i) When T_s is deterministic, let $G(t) = \begin{cases} 1, & t \geq T_s \\ 0, & t < T_s \end{cases}$, then from (10.2.6) we have

$$Q(T) \equiv (C_2 - C_1)m'(T) - \frac{\mathrm{d}p_c(T-T_s)}{\mathrm{d}T} = C_3 \qquad (10.2.7)$$

Assuming $p_c(T - T_s)$ to be increasing in $T_s < T \leq \infty$, we have $Q(\infty) < 0$ and $Q(T_s) = (C_2 - C_1)m'(T_s) > 0$. Furthermore, $Q(T)$ is always decreasing in $1/b < T \leq \infty$. Therefore, if $T_s \geq 1/b$ and $Q(T_s) > C_3$, there exists a finite and unique $T(T_0) > T_s$ satisfying (10.2.7) minimizing $C(T)$. Moreover there exists a unique $T = T_6(>T_0)$ satisfying $C(T) = C(T_s)$. If $Q(T_s) \leq C_3$, $\mathrm{d}C(T)/\mathrm{d}T > 0$, and (10.2.7) has no solution for $T > T_s$. Therefore $T_0 = T_s$ minimizes $C(T)$. If $T_s < 1/b$ and $Q(T_s) > C_3$, $Q(T)$ is decreasing in $T_s < T \leq \infty$, there exists a finite and unique $T(T_0) > T_s$ satisfying (10.2.7) minimizing $C(T)$. Also there exists a unique $T = T_6(>T_0)$ satisfying $C(T) = C(T_s)$. If $T_s < 1/b$ and $Q(T_s) \leq C_3$, and $Q(T)$ is deceasing in $T_s < T \leq \infty$, (10.2.7) has no solution for $T > T_s$. Therefore $T_0 = T_s$ minimizes $C(T)$. If $T_s < 1/b$, $Q(T_s) \geq C_3$ and $Q(T)$ is increasing in $(T_s, 1/b)$, there exists a finite and unique $T(T_0) > T_s$ satisfying (10.2.7) minimizing $C(T)$. If $T_s < 1/b$ and $Q(T_s) < C_3$, and $Q(T)$ is increasing $(T_s, 1/b)$ and the maximum value attained by $Q(T) \leq C_3$, $T_0 = T_s$ minimizes $C(T)$, and if maximum value attained by $Q(T) > C_3$ there exist two positive solutions $T = T_a$ and $T = T_b(T_s < T_a < T_b < \infty)$ satisfying

(10.2.7) $(\mathrm{d}^2C(T)/\mathrm{d}T^2 < 0|_{T=T_a}$ and $\mathrm{d}^2C(T)/\mathrm{d}T^2 > 0|_{T=T_b}$. Furthermore, if $C(T_b) < C(T_s)$ there exist two positive and unique $T = T_e$ and $T = T_f (T_s < T_e < T_b < T_f < \infty)$ satisfying $C(T) = C(T_s)$.

Moreover, for a specific operational time requirement $x \geq 0$ and reliability objective R_0 we have

$$R(x|T_s) = \mathrm{e}^{\left(-a\left((1+bT_s)\mathrm{e}^{(-bT_s)} - (1+b(T_s+x))\mathrm{e}^{-b(T_s+x)}\right)\right)},$$
$$R(x|\infty) = 1.$$

It is noted that $R'(x|T) < 0$ for $0 < T < T_x$ and $R'(x|T) > 0$ for $T_x < T < \infty$ where $T_x = \left(x\mathrm{e}^{-bx}\right)/\left(1 - \mathrm{e}^{-bx}\right)$. Therefore if $T_s \geq T_x$ and $R(x|T_s) < R_0$, there exists a unique $T(T_1) > T_s$ satisfying $R(x|T_s) = R_0, T > T_s$. If $T_s \geq T_x$ and $R(x|T_s) \geq R_0$ then $T_1 = T_s$ and if $T_s < T_x$ and $R(x|T_s) < R_0$, there exist a finite and unique $T(T_2) > T_s$ satisfying $R(x|T_s) = R_0, T > T_s$. If $T_s < T_x$ and $R(x|T_x) < R_0 < R(x|T_s)$ there exist two solutions $T = T_3$ and $T = T_4 (T_3 < T_x < T_4 < \infty)$ satisfying $R(x|T_s) = R_0$, $T > T_s$. Thus the optimal release policies minimizing the total expected testing cost subject to reliability requirement can be summarized as in Theorem 10.5.

Theorem 10.5 *Assuming* $C_2 > C_1 > 0$, $C_3 > 0$, $x \geq 0$, $0 < R_0 < 1$

(a) *If* $T_s \geq 1/b$ *and* $Q(T_s) > C_3$ *and*

(i) *If* $R(x|T_s) < R_0$, *then* $T^* = max\ (T_0, T_1)$
(ii) *If* $T_s \geq T_x$ *and* $R(x|T_s) \geq R_0$, *then* $T^* = T_0$
(iii) *If* $T_s < T_x$ *and* $R(x|T_s) = R_0$ *for* $T_0 \geq T_2$, $T^* = T_0$; *for* $T_0 < T_2$ *and* $T_2 > T_6$, $T^* = T_5$; *for* $T_0 < T_2$ *and* $T_2 < T_6$, $T^* = T_2$ *and for* $T_0 < T_2$ *and* $T_2 = T_6$, $T^* = T_s$ *or* T_2
(iv) *If* $T_s < T_x$ *and* $R(x|T_x) < R_0 < R(x|T_s)$, *for* $T_0 \leq T_3$ or $T_0 \geq T_4$, $T^* = T_0$; *for* $T_4 \geq T_6$, $T^* = T_3$; *for* $T_4 < T_6$ *if* $C(T_3) < C(T_4)$ *then* $T^* = T_3$ *and if* $C(T_3) > C(T_4)$ *then* $T^* = T_4$, *else* $T^* = T_3$ *or* T_4
(v) *If* $T_s < T_x$*and* $0 < R_0 \leq R(x|T_x)$ *then* $T^* = T_0$

(b) *If* $T_s \geq 1/b$ *and* $Q(T_s) \leq C_3$

(i) *If* $R(x|T_s) < R_0$ *then* $T^* = T_1$
(ii) *If* $R(x|T_s) \geq R_0$ *then* $T^* = T_s$

(c) *If* $T_s < 1/b$ *and* $Q(T_s) > C_3$ *and* $Q(T)$ *is decreasing in* (T_s, ∞), T^* *is obtained as in* (*a*) *above*.

(d) *If* $T_s < 1/b$ *and* $Q(T_s) \leq C_3$ *and* $Q(T)$ *is decreasing in* (T_s, ∞), T^* *is obtained as in* (*b*) *above*.

(e) *If* $T_s < 1/b$ *and* $Q(T_s) \geq C_3$ *and* $Q(T)$ *is increasing in* $(T_s, 1/b)$, T^* *is obtained as in* (*a*) *above*.

(f) *If* $T_s < 1/b$ *and* $Q(T_s) < C_3$, $Q(T)$ *is increasing in* $(T_s, 1/b)$ *and the maximum value reached by* $Q(T) \leq C_3$, T^* *is obtained as in* (*b*) *above*.

(g) *If* $T_s < 1/b$ *and* $Q(T_s) < C_3$, $Q(T)$ *is increasing in* $(T_s, 1/b)$ *and the maximum value reached by* $Q(T) > C_3$

(i) *If* $C(T_b) > C(T_s)$, *for* $R(x|T_s) < R_0$ *if* $T_1 < T_c$ *then* $T^* = T_1$, *else if* $T_c \le T_1 \le T_b$ *then* $T^* = T_b$ *and if* $T_1 > T_b$ *then* $T^* = T_1$ *and for* $R(x|T_s) \ge R_0, T^* = T_s$

(ii) *If* $C(T_b) = C(T_s)$
If $R(x|T_s) < R_0$ *then* $T^* = max\ (T_1, T_b)$
If $T_s \ge T_x$ *and* $R(x|T_s) \ge R_0$ *then* $T^* = T_b$
If $T_s < T_x$ *and* $R(x|T_s) = R_0$ *then for* $T_2 > T_b$, $T^* = T_s$; *for* $T_2 = T_b$, $T^* = T_s$ *or* T_b *and for* $T_2 < T_b$, $T^* = T_b$
If $T_s < T_x$ *and* $R(x|T_x) < R_0 < R(x|T_s)$ *then for* $T_b \le T_4$, $T^* = T_s$; *for* $T_b > T_4$ *if* $R(x|T_s) > R(x|T_b)$ *then* $T^* = T_s$ *else if* $R(x|T_s) < R(x|T_b)$ *then* $T^* = T_b$ *and if* $R(x|T_s) = R(x|T_b)$ *then* $T^* = T_s$ *or* T_b
If $T_s < T_x$ *and* $0 < R_0 \le R(x|T_x)$ *then for* $R(x|T_s) > R(x|T_b), T^* = T_s$ *for* $R(x|T_s) < R(x|T_b), T^* = T_b$ *and for* $R(x|T_s) = R(x|T_b), T^* = T_s$ *or* T_b

(iii) *If* $C(T_b) < C(T_s)$
If $R(x|T_s) < R_0$ *then* $T^* = max\ (T_b, T_1)$
If $T_s \ge T_x$ *and* $R(x|T_s) \ge R_0$ *then* $T^* = T_b$
If $T_s < T_x$ *and* $R(x|T_s) = R_0$ *then for* $T_2 > T_f$, $T^* = T_s$; *for* $T_2 = T_f$, $T^* = T_s$ *or* T_f; *for* $T_b < T_2 < T_f$, $T^* = T_2$ *and for* $T_s < T_2 \le T_b$, $T^* = T_b$
If $T_s < T_x$ *and* $R(x|T_x) < R_0 < R(x|T_s)$ *for* $T_b \le T_3$ *or* $T_b \ge T_4$, $T^* = T_b$; *for* $T_3 \le T_e$ *or* $T_4 \ge T_f$, $T^* = T_s$; *for* $T_3 > T_e$ *or* $T_4 \ge T_f$, $T^* = T_3$; *for* $T_3 \le T_e$ *or* $T_4 < T_f$, $T^* = T_4$ *and for* $T_3 > T_e$ *and* $T_4 < T_f$ *if* $C(T_3) < C(T_4)$ *then* $T^* = T_3$ *else if* $C(T_3) > C(T_4)$ *then* $T^* = T_4$ *and if* $C(T_3) = C(T_4)$ *then* $T^* = T_3$ *or* T_4
If $T_s < T_x$ *and* $0 < R_0 \le R(x|T_x)$ *then* $T^* = T_b$

Case (ii) When T_s has an arbitrary distribution $G(t)$ with finite mean μ then

$$P(T) \equiv (C_2 - C_1)\lambda_m(T) - \int_0^T \frac{\mathrm{d}p_c(T-t)}{\mathrm{d}t}\mathrm{d}G(t) = C_3 \tag{10.2.8}$$

Assuming $p_c(T - t)$ to be an increasing function in T for all $t(0 \le t \le T)$ we have $P(0) = (C_2 - C_1)\lambda_m(0) = 0$, $P(\infty) < 0$. It can be noted that $P(T)$ is a decreasing in $(1/b, \infty)$. Therefore, if $P(T)$ is decreasing in $(0, 1/b)$, (10.2.8) has no solution for $T > 0$, $\mathrm{d}C(T)/\mathrm{d}T > 0$ for all T and $T_0 = 0$ minimizes $C(T)$. If $P(T)$ is increasing in $(0, 1/b)$, the maximum value reached by $P(T)$ is $>C_3$. There exist two positive solutions $T = T_a$ and $T = T_b(0 < T_a < T_b < \infty)$ satisfying (10.2.8) $(\mathrm{d}^2C(T)/\mathrm{d}T^2 < 0|_{T=T_a}$ and $\mathrm{d}^2C(T)/\mathrm{d}T^2 > 0|_{T=T_b}$. Furthermore, if $C(T_b) < C(0)$

there exist $T_c > 0$ satisfying $C(T) = C(0)$. Moreover, for a specific operational time requirement $x \geq 0$ and reliability objective R_0 we have

$$R(x|0) = \mathrm{e}^{(-m(x))},$$
$$R(x|\infty) = 1.$$

It is noted that $R'(x|T) < 0$ for $0 < T < T_x$ and $R'(x|T) > 0$ for $T > T_x$ where $T_x = \left(x\mathrm{e}^{-bx}\right)/\left(1 - \mathrm{e}^{-bx}\right)$. Therefore if $R(x|0) < R_0 < 1$ there exist a unique $T(T_1) > 0$ satisfying $R(x|T) = R_0, T > 0$. If $R(x|T_x) < R_0 = R(x|0)$ there exist $T_2 \geq 0$ satisfying $R(x|T) = R_0, T > 0$. If $R(x|T_x) < R_0 < R(x|0)$ there exist T_3 and $T_4 (0 < T_3 < T_4)$ satisfying $R(x|T) = R_0$, $T > 0$.

Thus the optimal release policies minimizing the total expected cost subject to reliability requirement for this case can be summarized as in Theorem 10.6.

Theorem 10.6 *Assume $C_2 > C_1 > 0$, $C_3 > 0$, $x \geq 0$, $0 < R_0 < 1$, and μ is finite then the following apply*

(a) *If $P(T)$ is decreasing in $(0, \infty)$*

(i) *If $R(x|0) < R_0$ then $T^* = T_1$*
(ii) *If $R(x|0) \geq R_0$ then $T^* = 0$*

(b) *If $P(T)$ is increasing in $(0, 1/b)$ and the maximum value reached by $P(T) \leq C_3$*

(i) *If $R(x|0) < R_0$ then $T^* = T_1$*
(ii) *If $R(x|0) \geq R_0$ then $T^* = 0$*

(c) *If $P(T)$ is increasing in $(0, 1/b)$ and the maximum value reached by $P(T) > C_3$*

(i) *If $C(0) < C(T_b)$ and $R(x|0) < R_0$ then for $T_1 < T_c$, $T^* = T_1$; for $T_c \leq T_1 \leq T_b$, $T^* = T_b$ and for $T_1 > T_b$, $T^* = T_1$.*
(ii) *If $C(0) < C(T_b)$ and $R(x|0) \geq R_0$ then $T^* = 0$*
(iii) *If $C(0) = C(T_b)$ and $R(x|0) < R_0$ then $T^* = max\ (T_b, T_1)$*
(iv) *If $C(0) = C(T_b)$ and $R(x|0) = R_0 > R(x|T_x)$ then for $T_2 > T_b$, $T^* = 0$ and for $T_2 = T_b$, $T^* = 0$ or T_2*
(v) *If $C(0) > C(T_b)$ and $R(x|0) < R_0$ then $T^* = max\ (T_1, T_b)$*
(vi) *If $C(0) > C(T_b)$ and $R(x|0) = R_0 > R(x|T_x)$ then for $T_2 \leq T_b$, $T^* = T_b$; for $T_b < T_2 < T_f$, $T^* = T_2$; for $T_2 = T_f$, $T^* = 0$ or T_2 and for $T_2 > T_f$, $T^* = 0$*
(vii) *If $C(0) > C(T_b)$ and $R(x|T_x) < R_0 < R(x|0)$ then for $T_b \leq T_3$ or $T_b \geq T_4$, $T^* = T_b$; for $T_3 \leq T_c$ or $T_4 \geq T_f$, $T^* = 0$; for $T_3 > T_c$ or $T_4 \geq T_f$, $T^* = T_3$; for $T_3 \leq T_c$ or $T_4 < T_f$, $T^* = T_4$ and for $T_3 > T_c$ or $T_4 < T_f$ if $C(T_3) < C(T_4)$ then $T^* = T_3$; else if $C(T_3) > C(T_4)$ then $T^* = T_4$ and if $C(T_3) = C(T_4)$ then $T^* = T_3$ or T_4*
(viii) *If $C(0) > C(T_b)$ and $0 < R_0 \leq R(x|T_x)$ then $T^* = T_b$*

Application 10.2

We continue with the problem taken in Application 1. Using the same data set the parameters of s-shaped SRGM [42] are estimated to be $a = 103.984$ and $b = 0.265$. Let us assume the cost parameters to be $C_1 = \$10$, $C_2 = \$80$ and $C_3 = \$700$. If we assume the scheduled delivery time to be deterministic, define $p_c(T) = CT^2 = 150T^2$, and assume $R_0 = 0.87$, then the problem can be defined as

$$\begin{aligned} &\text{Minimize} \quad C(T) = 10m(T) + 80(m(T_l) - m(T)) + 700T + 150(T - T_s)^2 \\ &\text{Subject to} \quad R(x|T) \geq 0.87 \end{aligned} \tag{P6}$$

For a specific operational time requirement $x = 1$ week, scheduled delivery time $T_s = 30$ weeks from the time when testing starts and software life cycle time 3 years, it is estimated if testing is terminated in 20 weeks then the total testing and debugging cost incurred would be \$3268.74 and the achieved reliability level is 0.519. Now following Theorem 10.5 we obtain $T^* = T_s = 30$ weeks, the achieved level of reliability by the release time is $R^* = 0.9339$ and the total resources required are $C(T^*) = \$22062.81$.

Consider another case that when testing is started, only 4 weeks is remaining in the scheduled delivery, in this case the reliability achieved in 4 weeks of testing is estimated to be approximately zero, hence the software cannot be released at this time and software developer would have to pay the penalty cost to the user in order to achieve the reliability to the decided level of 0.87. So for this case again following Theorem 10.5 we obtain $T^* = T_s = 26.95$ weeks, $R^* = T_s = 0.87$ and $C(T^*) = \$21347.11$. Figures 10.4, 10.5 and 10.6 shows the cost curves and reliability growth curves for Application 10.2.

The release policies we have discussed up to now have been based on the time-dependent SRGM. The expected cost function includes the cost of testing per unit time. In reality testing cost increases with time and no software developer would spend infinite resources on testing the software, as instantaneous testing resources will decrease during the testing life cycle so that the cumulative testing effort approaches a finite limit [43]. The total testing effort expenditure never exceeds a predefined level (say α) even if the software is tested for an infinitely large time before release. In the literature many SRGM have been developed which describe the growth of testing process with respect to the testing efforts spent. If we use such an SRGM to formulate the cost model of the release time problem then the testing cost can be defined per unit testing effort expenditure. In the next section we will discuss the release policy based on a testing effort dependent SRGM.

10.2.3 Release Policy Based on Testing Effort Dependent SRGM

Kapur and Garg [6] discussed release policies using exponential, modified exponential and S-shaped test effort based SRGM for maximizing expected gain

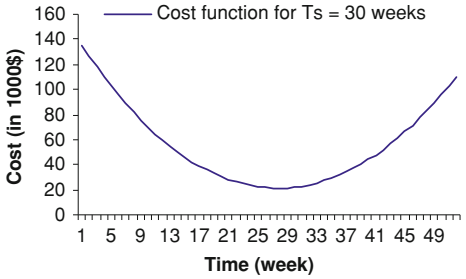


Fig. 10.4 Cost function of Application 10.2 for $T_s = 30$ weeks

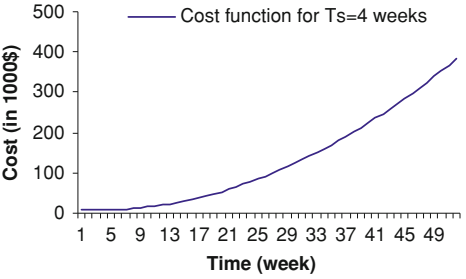


Fig. 10.5 Cost function of Application 10.2 for $T_s = 4$ weeks

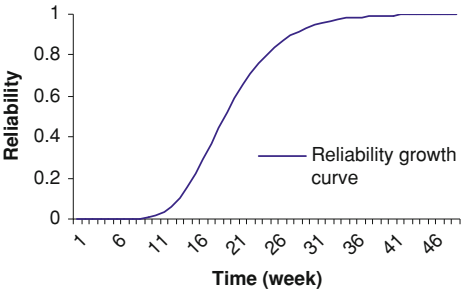


Fig. 10.6 Reliability growth curve for Application 10.2

function subject to achieving a given level of failure intensity. Mathematically stated as

$$\begin{aligned} &\text{Maximize} \quad G(T) = (C_2 - C_1)m(T) - C_3 W(T) \\ &\text{Subject to} \quad \lambda(T) \leq \lambda_0 \end{aligned} \tag{P7}$$

where C_3 is now the expected cost per unit testing effort expenditure and $m(t)$ is the mean value function of a test effort based SRGM (see Sect. 2.7). In the previous sections of the chapter we have discussed the release policies for exponential and s-shaped SRGM. Now in this section we choose the modified exponential SRGM [41, 43] to formulate the release policy. The mean value function of the test effort based modified exponential model is given as

$$m(t) = \sum_{i=1}^{2} m_i(t) = a \sum_{i=1}^{2} r_i \left(1 - \mathrm{e}^{-b_i W(t)}\right)$$

and the failure intensity function is given as

$$\lambda(t) = aw(t) \sum_{i=1}^{2} r_i b_i \left(1 - \mathrm{e}^{-b_i W(t)}\right) \tag{10.2.9}$$

where $W(t)$ is the distribution of the testing effort and can be described by exponential, Rayleigh, Weibull, logistic, etc. curves (see Sect. 2.7). Hence the release policy for this SRGM can be rewritten as

$$\begin{aligned} &\text{Maximize} \quad G(T) = \sum_{i=1}^{2} (C_{2i} - C_{1i}) m_i(T) - C_3 W(T) \\ &\text{Subject to} \quad \lambda(T) \le \lambda_0 \end{aligned} \tag{P8}$$

The first derivative of the gain function is zero when

$$a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i \mathrm{e}^{-b_i W(T)} = C_3 \tag{10.2.10}$$

Hence if $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i > C_3$ and $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i \mathrm{e}^{-b_i W(T)} < C_3$ there exists a finite and unique $T = T_0 (0 < T < T_l)$ satisfying (10.2.10). If $G'(T) < 0$ for $T > 0, a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i \le C_3$, and if $G'(T) > 0$ for $T < T_l$, $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i e^{-b_i W(T)} \ge C_3$. From (10.2.9) it may be observed that either $\lambda(t)$ is decreasing in $T (0 \le T \le T_l)$ or is increasing in $(0, t_x)$ and decreasing in (t_x, T_l) where $T = t_x$ satisfies $\lambda'(t) = 0$. Thus when $\lambda(t)$ is decreasing in $T (0 \le T \le T_l)$, $\lambda(0) > \lambda_0$ and $\lambda(T_l) \le \lambda_0$, there exists a finite and unique $T = T_1 (\le T_l)$ satisfying $\lambda(T) = \lambda_0$. If $\lambda(t)$ is increasing in $(0, t_x)$ and decreasing in (t_x, T_l), $\lambda(0) > \lambda_0$ and $\lambda(T_l) \le \lambda_0$ there exists a finite and unique $T = T_1 (T_x < T_1 \le T_l)$ satisfying $\lambda(T) = \lambda_0$. Combining the gain and intensity requirements, we may state the following theorem for optimal release policy.

Theorem 10.7 *Assume* $C_{2i} > C_{1i} > 0 (i = 1, 2)$, $C_3 > 0$, $\lambda_0 > 0$ *and* $\lambda(T_l) \le \lambda_0$.

(a) *If* $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i > C_3$ *and* $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i \mathrm{e}^{-b_i W(T_l)} < C_3$ *then*

(i) *when* $\lambda(T)$ *is decreasing in* T,
$T^* = max\ (T_0, T_1)$, *for* $\lambda(0) > \lambda_0$ *or* $T^* = T_0$ *for* $\lambda(0) \le \lambda_0$

(ii) *when* $\lambda(T)$ *is increasing in* $(0, t_x)$ *and decreasing in* (t_x, T_l)
$T^* = max\ (T_0, T_1)$, *for* $\lambda(t_x) > \lambda_0$ *or* $T^* = max\ (T_0, t_x)$ *for* $\lambda(t_x) \le \lambda_0$

(b) *If* $a \sum_{i=1}^{2} (C_{2i} - C_{1i}) r_i b_i \le C_3$ *then*

(i) *when* $\lambda(T)$ *is decreasing in* T,
$T^* = T_1$, *for* $\lambda(0) > \lambda_0$ *or* $T^* = 0$ *for* $\lambda(0) \le \lambda_0$

(ii) *when* $\lambda(T)$ *is increasing in* $(0, t_x)$ *and decreasing in* (t_x, T_l)
$T^* = T_1$, *for* $\lambda(t_x) > \lambda_0$ *or* $T^* = t_x$ *for* $\lambda(t_x) \leq \lambda_0$

(c) *If* $a\sum_{i=1}^{2}(C_{2i} - C_{1i})r_i b_i e^{-b_i W(T_l)} \geq C_3$ *then* $T^* = T_1$.

It may be noted that if $\lambda(T_l) > \lambda_0$, software may not be released as the failure intensity constraint has not been met. In such a situation more testing resources may be needed to achieve the desired failure intensity before releasing the software.

Application 10.3

We again continue with the problem taken in Application 10.1. Using the same data set the parameters of the modified exponential SRGM are estimated. First an exponential test effort function $W(t) = \alpha(1 - e^{-\beta t})$ is chosen to describe the testing effort expenditure and its parameters are estimated to be $\alpha = 35{,}390$ CPU hours and $\beta = 0.017$. Using the estimates of testing effort function the parameters of the SRGM are estimated to be $a = 120.03$, $r_1 = 0.425$, $r_2 = 0.575$, $b_1 = 0.000178$ and $b_2 = 0.00018$. Let us assume the cost parameters $C_{11} = \$100$, $C_{12} = \$150$, $C_{21} = \$1{,}500$, $C_{22} = \$1{,}700$ and $C_3 = \$5$. Let the software life cycle be $T_l = 200$ weeks. Now if the software is release untested, the failure intensity of the software is $\lambda(0) = 1.309$ which is of a high level and the reliability of the software at that time would be negligible and the value of gain function would be \$15243.34. When testing is continued only up to the time for which data are available (20 weeks) in that case failure intensity would be very high, $\lambda(20) = 7.809$ (note that failure intensity function first increases and then decreases in this case) and reliability $R(20) = 0.654$ and the value of gain function would be \$99132.01. Although the gain function is reaching near its peak at this time [peak value is $(G(20.27) = \$99137.90]$, the software cannot be released at this time as the failure intensity is very high. The failure intensity has its peak at $T = 18.67$, it is increasing before this time and decreasing later. Now we apply the release policy (P8) to find the optimal time to release for $\lambda_0 = 0.8$ then the optimal solution obtained following Theorem 10.7 is $T^* = 164.03$ weeks. At this time $R(T^*) = 0.9933$, and $G(T^*) = 11883.55$. If we further decrease the failure intensity requirement to $\lambda_0 = 0.6$ in that case $T^* = 180.96$ weeks. At this time $R(T^*) = 0.9939$ and $G(T^*) = \$9202.81$. Graphical plots of gain and failure intensity functions are shown in Figs. 10.7 and 10.8, respectively.

10.2.4 Release Policy for Random Software Life Cycle

Yun and Bai [8] proposed that software release time problems should assume software life cycle to be random as several factors such as availability of alternative competitive product in the market, a better product announcement by the developer himself, etc. plays a role in the determination of the software life cycle length. They obtained the optimal release time solutions numerically using bisection method for exponential, modified exponential and S-shaped distribution,

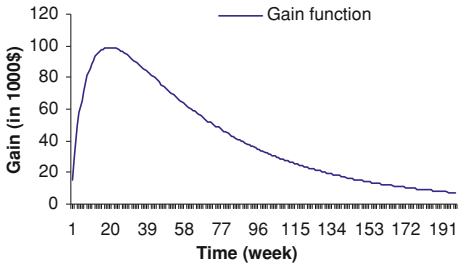


Fig. 10.7 Gain function for Application 10.3

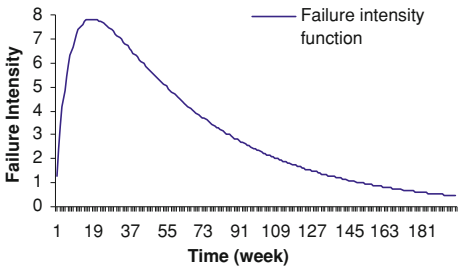


Fig. 10.8 Failure intensity function for Application 10.3

maximizing the total average profit with random life cycle. Later Kapur et al. [10] determined release policies for a software system based on minimizing expected cost subject to achieve a desired level of intensity assuming software life cycle to be random. To describe the software release time policy with random life cycle assume that $h(t)$, $H(t)$ and $r(t)$ be the *pdf*, *cdf* and hazard rate of the software life cycle length. The expected software cost during the software life cycle (if the software is released at time T) is

$$\begin{aligned} C(T) = C_1 \int_0^T m(t)h(t)\mathrm{d}t + C_3 \int_0^T th(t)\mathrm{d}t + C_1 \int_T^\infty m(T)h(t)\mathrm{d}t \\ + C_3 T\overline{H}(T) + C_2 \int_T^\infty (m(t) - m(T))h(t)\mathrm{d}t \end{aligned} \tag{10.2.11}$$

Hence if we state the problem as minimizing the expected software cost subject to the failure intensity requirement $\lambda(T) \leq \lambda_0$, simplifying the cost function is stated as

$$\begin{aligned} \text{Minimize} \quad C(T) = C_1 \int_0^T m(t)h(t)\mathrm{d}t + C_3 \int_0^T th(t)\mathrm{d}t + (C_1 - C_2)m(T)\overline{H}(T) \\ + C_3 T\overline{H}(T) + C_2 \int_T^\infty m(t)h(t)\mathrm{d}t \end{aligned}$$

$$\text{Subject to} \quad \lambda(T) \leq \lambda_0 \tag{P9}$$

The conditional failure intensity at T is given as $I(T) = \int_T^{\infty} m'(T)h(t)\mathrm{d}t, I(T)$ is decreasing in T, if $I(0) > \lambda_0$ there exists a finite and unique $T(T_1)$ satisfying $I(T)|_{T=T_1} = \lambda_0$. Combining the cost and intensity requirements the theorem obtained is

Theorem 10.8

1. *If* $ab \leq \frac{C_3}{(C_2 - C_1)}$ *and* $I(0) \leq \lambda_0$, *then* $T^* = 0$.
2. *If* $ab \leq \frac{C_3}{(C_2 - C_1)}$ *and* $I(0) > \lambda_0$, *then* $T^* = T_1$
3. *If* $ab > \frac{C_3}{(C_2 - C_1)}$ *and* $I(0) \leq \lambda_0$, *then* $T^* = T_0$
4. *If* $ab > \frac{C_3}{(C_2 - C_1)}$ *and* $I(0) > \lambda_0$, *then* $T^* = max\ (T_0, T_1)$

where T_0 is the time as described in Theorem 10.1.

Application 10.4

Consider the problem discussed in Application 10.1. Since this policy is also discussed for the Goel and Okumoto [39] exponential SRGM we have $a = 130.30$ and $b = 0.083$. Let us again assume the cost parameters to be $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$500$. If we assume the *pdf* of the software life cycle length $h(t) = 0.005 \exp(-208t)$ and if testing is continued for 20 weeks only and then terminates, the total expected cost that would be incurred in the software testing is \$5991.04 and the achieved failure intensity would be $I(20) = 1.943$, which is not an acceptable level if we want to achieve failure intensity less than or equal to 0.1. Now we apply Theorem 10.8 to obtain the release policy for this case. It is computed that the optimal release time is $T^* = 53.78$ weeks and $C(T^*) = \$20740.55$. At this time $R(T^*) = 0.8873$. Graphical plots of cost and failure intensity functions are shown in Figs. 10.9 and 10.10, respectively.

10.2.5 A Software Cost Model Incorporating the Cost of Dependent Faults Along with Independent Faults

Often in the testing process the software debugging team removes some additional faults while debugging a fault that has caused the failure. The faults which are removed on the go are called dependent faults and those which become the

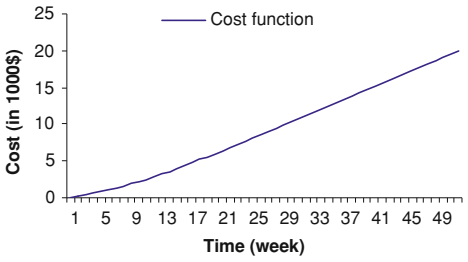


Fig. 10.9 Cost function for Application 10.4

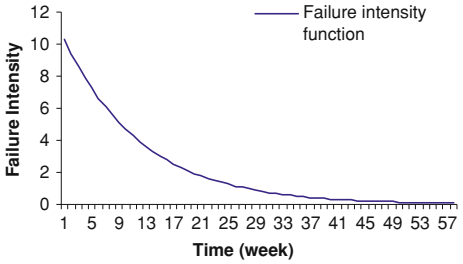


Fig. 10.10 Failure intensity function for Application 10.4

primary cause of a failure are called independent faults in the literature. Most of the release policies that have been discussed so far in this chapter or literature ignore the removal of dependent faults. Although there is no testing cost incurred for the detection and removal of these faults there is some additional removal cost associated with these removals. In this section we are addressing one such release policy. The formulation of such a policy requires an SRGM that accounts for the removal of dependent faults along with the independent faults. Kapur and Garg [7] SRGM for error removal phenomenon (refer to Sect. 2.3.7) describes this aspect of the testing process. The authors in the original article have proposed this SRGM and formulated the release policy using the model. They modified the simple cost model to include the cost of additional removals. Note that in this case there are more removals as compared to the failures and at any time t, the number of dependent faults removed is given by the difference of mean number of removals and failures, i.e. $m_r(t) - m_f(t)$.

Now if we assume $C_1^{'}$ is the cost removal of a dependent fault in the testing phase and C_2' is the corresponding cost for the operational phase then the simple cost model can be modified as

$$\begin{aligned} C(T) = {} & C_1 m_f(T) + C_1'\big(m_r(T) - m_f(T)\big) + C_2\big(m_f(T_l) - m_f(T)\big) \\ & + C_2'\big(m_r(T_l) - m_r(T) - \big(m_f(T_l) - m_f(T)\big)\big) + C_3 T \end{aligned} \tag{10.2.12}$$

Using the cost model (10.2.12) the release policy is stated as

$$\begin{aligned} \text{Minimize} \quad & C(T) = C_1 m_f(T) + C_1'\big(m_r(T) - m_f(T)\big) + C_2\big(m_f(T_l) - m_f(T)\big) \\ & \qquad + C_2'\big(m_r(T_l) - m_r(T) - \big(m_f(T_l) - m_f(T)\big)\big) + C_3 T \\ \text{Subject to} \quad & R(x|T) \geq R_0 \end{aligned} \tag{P10}$$

The optimal release policies are obtained by differentiating the expected cost function with respect to time and equating to zero, i.e., $C'(T) = 0$. We have $C'(T) = 0$ if

$$m_r'(T) + (D-1)m_f'(T) = \frac{C_3}{C_2' - C_1'} \tag{10.2.13}$$

where $D = \frac{C_2 - C_1}{C_2' - C_1'}$. It may be observed that if $pD \geq q$ and $apD > \frac{C_3}{C_2' - C_1'}$ (p and q are the parameters of the SRGM), finite and unique $T = T_0 (>0)$ exists, satisfying Eq. (10.2.13). If $pD \geq q$ and $apD \leq \frac{C_3}{C_2' - C_1'}$, $C'(T) > 0$ for $T > 0$ and $T = 0$ minimizes $C(T)$. If $pD < q$, $C'(T) > 0$ and $m_r'(T_m) + (D-1)m_j'(T_m) \leq \frac{C_3}{C_2' - C_1'}$ for $T > 0$, or if $pD < q$, and $apD \geq \frac{C_3}{C_2' - C_1'}$ a finite and unique $T = T_2 (> T_m)$ exists, satisfying Eq. (10.2.13). On the other hand if $pD < q$, $apD < \frac{C_3}{C_2' - C_1'}$ and $m_r'(T_m) + (D-1)m_j'(T_m) > \frac{C_3}{C_2' - C_1'} T = T_y$ and $T = T_z (0 < T_y < T_z)$ exists, satisfying Eq. (10.2.13). It may be noted that $C''(T) < 0$ for $T = T_y$ and $C''(T) > 0$ for $T = T_z$. Moreover, if $C(0) < C(T_z)$, a finite and unique $T = T_c (< T_y)$ exists, satisfying $C(T) = C(T_z)$; whereas if $C(0) > C(T_z)$ finite and unique $T = T_c$ and $T = T_f$ exist, satisfying $C(T) = C(0)$. Combining the cost and reliability requirements the release time can be determined according to the Theorem 10.9.

Theorem 10.9 *Assume* $C_1 > C_1' > 0$, $C_2 > C_1$, $C_2 > C_2' > C_1'$, $C_3 > 0$, $x > 0$, $0 < R_0 < 1$ *states that*

(a) *when* $aD \geq b$,

(i) $NaD > \frac{C_3}{C_2' - C_1'}, T^* = max(T_0, T_1)$ *for* $R(x|0) < R_0 < 1$, *or* $T^* = T_0$ *for* $R(x|0) \geq R_0 > 0$.

(ii) $NaD \leq \frac{C_3}{C_2' - C_1'}, T^* = T_1$ *for* $R(x|0) < R_0 < 1$, *or* $T^* = 0$ *for* $R(x|0) \geq R_0 > 0$.

(b) *when* $aD < b$,

(i) $m_r'(T_m) + (D-1)m_j'(T_m) \leq \frac{C_3}{C_2' - C_1'}, T^* = T_1$ *for* $R(x|0) < R_0 < 1$, *or* $T^* = 0$ *for* $R(x|0) \geq R_0$.

(ii) $NaD \geq \frac{C_3}{C_2' - C_1'}, T^* = max(T_2, T_1)$ *for* $R(x|0) < R_0 < 1$, *or* $T^* = T_2$ *for* $R(x|0) \geq R_0 > 0$.

(iii) $NaD < \frac{C_3}{C_2' - C_1'}$ *and* $m_r'(T_m) + (D-1)m_f'(T_m) > \frac{C_3}{C_2' - C_1'}$
If $C(0) = C(T_z) T^* = max\ (T_z, T_1)$ *for* $R(x|0) < R_0 < 1$, *or*
$T^* = 0$ *or* T_z *for* $R(x|0) \geq R_0 > 0$.
else if $C(0) < C(T_z) T^* = 0$ *or* T_z *for* $R(x|0) \geq R_0 > 0$, *or*
else if $C(0) < C(T_z)$ *and* $R(x|0) < R_0 < 1$ $T^* = T_1$ *for* $T_1 < T_c$
$T^* = T_c$ *or* T_z *for* $T_1 = T_c$; $T^* = max\ (T_z, T_1)$ *for* $T_1 > T_c$
and if $C(0) > C(T_z) T^* = max\ (T_z, T_1)$ *for* $R(x|0) < R_0 < 1$, *or*
$T^* = T_z$ *for* $R(x|0) \geq R_0 > 0$.

Here T_0 *and* T_1 *are as defined in Theorems* 10.1 *and* 10.2.

Application 10.5

For application of the above release policy we again continue with the data set chosen in Application 10.1. First we estimate the parameters of the Kapur and

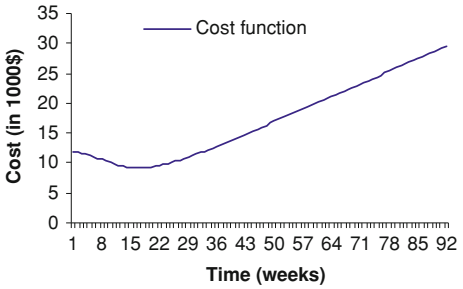


Fig. 10.11 Cost function for Application 10.5

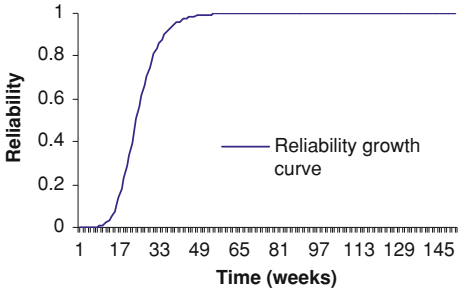


Fig. 10.12 Reliability growth function for Application 10.5

Garg [7] software reliability growth model using these data. The estimated values of the parameters of the SRGM are $a = 273.825$, $p = 0.035$ and $q = 0.136$. Let the cost parameters be $C_1 = \$10$, $C'_1 = \$5$, $C_2 = \$50$, $C'_2 = \$40$ and $C_3 = \$300$. The operational mission time is $x = 1$ week, the software life cycle length is 3 years (156 weeks). The reliability level achieved in 20 weeks of testing (time period for which data are available) is 0.2836 and the total testing cost in software life cycle for this level of reliability would be \$43324.47. Now using the available information in Theorem 10.9 we obtain the optimal release time of the software, $T^* = 32.55$ weeks for a reliability requirement of $R_0 = 0.85$ and $C(T^*) = \$11874.76$. Graphical plots of cost and reliability growth functions are shown in Figs. 10.11 and 10.12, respectively.

10.2.6 Release Policies Under Warranty and Risk Cost

The focal role of cost function in determining the optimal release time of software enforced [14] to further modify the cost function by incorporating warranty and risk costs associated with the testing life cycle. Authors claimed that the different faults take different times for their removal and for this reason associated the time factor with the cost of fault removal. The cost model (10.2.1) is modified and total expected software cost is defined as

$$E(T) \equiv C(T) = C_0 + C_1 m(T)\mu_y + C_3 T^{\alpha} + C_4 \mu_W (m(T + T_W) - m(T)) + C_5 (1 - R(x|T)) \tag{10.2.14}$$

Now we explain each term in the cost function. Here C_0 is defined as the fixed setup cost of testing. Under the assumption that it takes time to remove faults and removal time of each fault follows a truncated exponential distribution the probability density function of the time to remove a fault during testing period, Y, is given by

$$s(y) = \begin{cases} \frac{\lambda_y e^{-\lambda_y y}}{\int_0^{T_0} \lambda_y e^{-\lambda_y x}} & \text{for } 0 \le y \le T_0 \\ 0 & \text{for } y > T_0 \end{cases}$$

where λ_y is a constant parameter associated with truncated exponential density function Y and T_0 is the maximum time to remove any error during testing period. Then the expected time to remove each error is given by

$$E(Y) = \mu_y = \int_0^{T_0} y s(y) \mathrm{d}y = \frac{1 - (\lambda_y T_0 + 1) e^{-\lambda_y T_0}}{\lambda_y (1 - e^{-\lambda_y T_0})}$$

Hence, the expected total time to remove $N(T)$ faults corresponding to all failures experienced up to time T is given by

$$E\left(\sum_{i=1}^{N(T)} Y_i\right) = E(N(T)) \cdot E(Y_i) = m(T) \cdot \mu_y$$

Thus, the expected cost to remove all errors detected by time T in the testing phase, where C_1 is now the cost of removing an error per unit time during testing phase is given by

$$E_1(T) = C_1 E\left(\sum_{i=1}^{N(T)} Y_i\right) = C_1 m(T)\mu_y \tag{10.2.15}$$

The cost of testing per unit time is assumed to be a power function of time T since the cost of testing increases with higher gradient in the beginning and slows down later.

$$E_2(T) = C_3 T^{\alpha} \tag{10.2.16}$$

Further it is assumed that the software developer does not maintain the software for the whole of its operational life cycle. This is because the software developers always keep on improving their software and come up with newer versions with added features and improved reliability. The newer versions are usually launched even before the earlier version obsoletes and the developer encourages the users of the previous versions to improve their version with the new one as it has enhanced features. So given any version of the software, developer decides a warranty period

for which they provide aftersales services and after that period if a failure is encountered no removal is made from the part of the developer. Hence now instead of calculating the cost for the whole life cycle of the software we need to calculate it only up to the time when the warranty period ends. The expected cost to remove all faults during warranty period $[T,\ T + T_w]$ is given by

$$E_3(T) = C_4\mu_W(m(T + T_W) - m(T)) \tag{10.2.17}$$

A new cost i.e. risk cost of failure in the operational phase is also added to the expected cost function. Consideration of risk cost is an important attribute for the complex software systems, which are designed for implementation in critical system environments and applications. Failure in critical systems can result in huge risk cost to software developers hence long run testing and very high level of reliability are desired for these systems. The risk cost due to software failure after releasing the software can be expressed as

$$E_4(T) = C_5(1 - R(x|T)) \tag{10.2.18}$$

The optimal release time is determined minimizing the unconstrained expected total development software cost function for the Goel and Okumoto [39] exponential SRGM. The policy can be stated as

$$\begin{aligned}\text{Minimize}\quad E(T) \equiv C(T) &= C_0 + C_1 m(T)\mu_y + C_3 T^{\alpha} \\ &\quad + C_4\mu_W(m(T + T_W) - m(T)) + C_5(1 - R(x|T))\end{aligned} \tag{P11}$$

Release time is determined by differentiating $E(T)$ with respect to time, i.e.

$$\begin{aligned}E'(T) = \alpha C_3 T^{\alpha-1} &- C_4\mu_W ab\mathrm{e}^{-bT}\left(1 - \mathrm{e}^{-bT_W}\right) \\ &- ab\mathrm{e}^{-bT}\left(C_5\left(1 - \mathrm{e}^{-bx}\right)R(x|T) - C_1\mu_y\right)\end{aligned}$$

The second derivative of $E(T)$ with respect to time is

$$E''(T) = \mathrm{e}^{-bT}\begin{pmatrix}\alpha(\alpha-1)C_3T^{\alpha-2}\mathrm{e}^{bT} + C_4\mu_W ab^2\left(1 - \mathrm{e}^{-bT_W}\right) - C_1\mu_y ab^2 \\ + C_5 ab^2\left(1 - \mathrm{e}^{-bx}\right)R(x|T)\left(1 - a\mathrm{e}^{-bT}\left(\left(1 - \mathrm{e}^{-bx}\right)\right)\right)\end{pmatrix}$$

Equivalently it can be written as

$$E''(T) = \mathrm{e}^{-bT}(u(T) - C)$$

where

$$u(T) = \alpha(\alpha-1)C_3T^{\alpha-2}\mathrm{e}^{bT} + C_5 ab^2\left(1 - \mathrm{e}^{-bx}\right)R(x|T)\left(1 - a\mathrm{e}^{-bT}\left(\left(1 - \mathrm{e}^{-bx}\right)\right)\right)$$

and $C = C_1\mu_y ab^2 - C_4\mu_W ab^2\left(1 - e^{-bT_W}\right)$. Since $u'(T) > 0$, $u(T)$ is an increasing function of time. The following theorem gives the optimal value of release time T^* minimizing the expected total cost of the software.

Theorem 10.10 *Given the values* C_1, C_3, C_4, C_5, x, μ_y, μ_w, T_w.

(a) *If* $u(0) \geq C$ *then* $u(T) \geq C$ *for any* T *and*

(i) *If* $E'(0) \geq 0$, *then* $T^* = 0$.
(ii) *If* $E'(\infty) < 0$, then $T* = \infty$.
(iii) *If* $E'(0) < 0$, *then there exist a* T' *such that* $E'(T) < 0$, *for any* $T \in (0, T']$ *and* $E'(T) > 0$, *for any* $T \in (T', \infty)$ *hence* $T^* = T'$.

(b) *If* $\mu(\infty) < C$ *then* $u(T) \leq C$ *for any* T *and*

(i) *If* $E'(0) \leq 0$, *then* $T^* = \infty$.
(ii) *If* $E'(\infty) > 0$, *then* $T^* = 0$.
(iii) *If* $E'(0) > 0, E'(\infty) < 0$, *then there exist a* T'' *such that* $E'(T) > 0$, *for any* $T \in (0, T'']$ *and* $E\prime(T) < 0$, *for any* $T \in (T'', \infty)$, *then* $T^* = 0$ *if* $E(0) \leq E(\infty)$ *and* $T^* = \infty$ *if* $E(0) > E(\infty)$ *where* $T'' = E'^{-1}(0)$

(c) *If* $u(0) < C$, $\mu(\infty) > C$ *then there exist a* T^0 *such that* $u(T) < C$ *for* $T \in (0, T^0]$ *and* $u(T) > C$ *for* $T \in (T^0, \infty)$ *where* $T^0 = u^{-1}(C)$, *then*

(i) *If* $E'(0) \geq 0$, *then* $T^* = 0$
(ii) *If* $E(0) \leq E(T_b)$ *and* $T^* = T_b$
(iii) *If* $E(0) > E(T_b)$ *where* $T_b = inf\{T > T_b : E'(T) > 0\}$
(iv) *If* $E'(0) < 0$, *then* $T^* = T_b''$, *where* $T_b'' = E'^{-1}(0)$.

Application 10.6

For application of the above release policy let us consider the data set reported by Musa et al. [44] based on the failures from a real time command and control system, which represents the failures observed during system testing for 25 CPU hours. During this time period 136 faults have been discovered. The delivery number of object instructions for this system was 21,700 and was developed by Bell Laboratories. Using this data set the parameters of Goel and Okumoto [39] SRGM are estimated to be $a = 142.32$ and $b = 0.1246$. if the cost parameters are $C_0 = \$50$, $C_1 = \$60$, $C_3 = \$700$, $C_4 = \$3{,}600$ and $C_5 = \$50{,}000$. The cost $C_5 = \$50{,}000$ is usually very high as it represents the risk cost of field failures and includes the cost of loss of revenue, customers and even the human life. If the operational mission time $x = 1$ CPU hour, the warranty period length is $T_w = 450$ CPU hours, $\alpha = 0.95$, $\mu_w = 0.5$ and $\mu_y = 0.1$ then if testing is stopped after 25 CPU hours of testing, the total cost incurred would be \$53275.42 and the reliability would reach to the level of 0.4772. Now if we apply Theorem 10.10 to determine the release time we obtain $T^* = 43.76$ CPU hours, the total cost incurred would be $C(T^*) = \$30{,}804$ and in this time period the software will

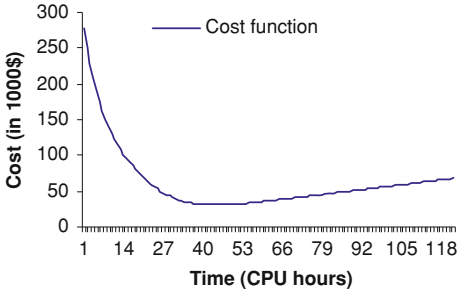


Fig. 10.13 Cost function for Application 10.6

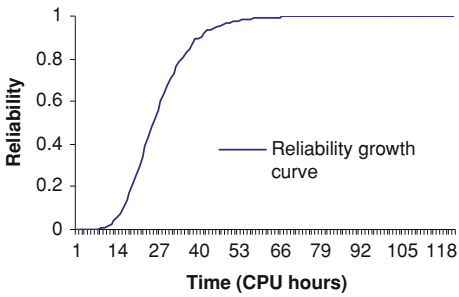


Fig. 10.14 Reliability growth function for Application 10.6

become 93.104% reliable. Graphical plots of cost and reliability growth functions are shown in Figs. 10.13 and 10.14, respectively.

10.2.7 Release Policy Based on SRGM Incorporating Imperfect Fault Debugging

In Chap. 3 of the book we have discussed a number of testing efficiency based SRGM clearly stating the need and importance of incorporating testing efficiency parameters in the SRGM. So it is equally important to formulate the release policies based on imperfect debugging models. Kapur and Garg [5] made an initial attempt in introducing the concept of imperfect fault debugging in NHPP based SRGM, assuming fault removal rate per remaining faults is reduced due to imperfect fault debugging (see Sect. 3.3.1). They have also discussed the release policy for the SRGM minimizing the total expected software cost subject to software reliability not less than a specified reliability objective. The simple cost model (10.2.1) is modified to include separate cost of fixing a fault due to perfect and imperfect fault debugging during testing and operational phases along with the testing cost per unit time. Defining p, as the probability of perfect debugging, the cost function is redefined as

$$C(T) = \big(C_1 p + C_1'(1-p)\big)m_f(T) + \big(C_2 p + C_2'(1-p)\big)\big(m_f(T_l) - m_f(T)\big) + C_3 T \tag{10.2.19}$$

Using the cost function (10.2.12) the release policy is stated as

$$\begin{aligned} \text{Minimize} \quad & C(T) = \big(C_1 p + C_1'(1-p)\big) m_f(T) \\ & \quad + \big(C_2 p + C_2'(1-p)\big)\big(m_f(\infty) - m_f(T)\big) + C_3 T \qquad \text{(P12)} \\ \text{Subject to} \quad & R(x|T) \geq R_0 \end{aligned}$$

The optimal release policy is determined using the principles of calculus and combining the cost and reliability requirements

Theorem 10.11 *Assuming* $C_2 > C_1 > 0$, $C_2' > C_1' > 0$, $C_3 > 0$, $x > 0$, $0 < R_0 \leq 1$, $D_1 = C_1 p + C_1'(1-p)$ *and* $D_2 = C_2 p + C_2'(1-p)$

(a) *If* $ab > \frac{C_3}{(D_2 - D_1)}$ *and* $R(x|0) < R_0 < 1$ *then* $T^* = max\ (T_0, T_1)$

(b) *If* $ab > \frac{C_3}{(D_2 - D_1)}$ *and* $R(x|0) \geq R_0 > 0$ *then* $T^* = T_0$

(c) *If* $ab \leq \frac{C_3}{(D_2 - D_1)}$ *and* $R(x|0) < R_0 < 1$ *then* $T^* = T_1$

where T_1 and T_0 are as defined in Theorems 10.1 and 10.2.

Application 10.7

Let us again consider the data set considered in Application 10.6. Using this data set the parameters of Kapur and Garg [5] imperfect fault debugging SRGM are estimated to be $a = 126.39$, $b = 0.154$ and $p = 0.903$. Let the cost parameters $C_1 = \$200$, $C'_1 = \$110$, $C_2 = C'_2 = \$1{,}500$ and $C_3 = \$50$. If the operational mission time $x = 1$ CPU hour, the software life cycle length is $T_l = 2{,}920$ CPU hours, then if testing is stopped after 25 CPU hours of testing, the total cost incurred would be \$33684.21 and the reliability would reach to the level of 0.5702. According to Theorem 10.11 the optimal release time for the software is $T^* = 44.82$ weeks for the reliability requirement of 0.85. The optimal cost value is $C(T^*) = \$29372.21$ with the achieved reliability of 0.9649. The optimal solution yields more reliability then the aspiration level as the case considered here is part 1 of Theorem 10.11, $T^* = \max\ (T_0, T_1) = T_0$ (point of minima on cost curve).

Xie and Yang [45] also determined the optimal release time policy based on pure imperfect fault debugging SRGM proposed by Kapur and Garg [5] though they referred it as the SRGM proposed by Obha and Chou [46] on error generation. Authors claimed that the cost of testing C_3 is a function of perfect debugging probability p, since the testing cost parameter depends on the testing team composition and testing strategy used. If the probability of perfect debugging is to be increased, it is expected that extra financial resources will be needed to engage more experienced testing personnel, and that will result in an increase in C_3. The modified cost model is given as

$$C(T, p) = C_1 m(T) + C_2(m(T_l) - m(T)) + \frac{C_3 T}{(1-p)} \qquad (10.2.20)$$

Then the optimal release time T and optimal testing level p are determined minimizing the cost function. Since the SRGM is wrongly assumed to be of error

generation type, the cost of imperfectly removing error was not included in the cost function.

Kapur et al. [47] modified the above cost model, incorporating separate cost of fixing an error due to perfect and imperfect fault debugging during testing and operational phases

$$C(T,p) = \big(C_1 p + C_1'(1-p)\big)m_f(T) + \big(C_2 p + C_2'(1-p)\big)\big(m_f(T_l) - m_f(T)\big) + \frac{C_3 T}{(1-p)} \tag{10.1.21}$$

The optimal release policy minimizing the total expected software at optimal testing level p^* is formulated as

$$\text{Minimize} \quad C(T,p) = \big(C_1 p + C_1'(1-p)\big)m_f(T) + \big(C_2 p + C_2'(1-p)\big)\big(m_f(T_l) - m_f(T)\big) + \frac{C_3 T}{(1-p)} \quad \text{(P13)}$$

$$\text{Subject to} \quad 0 < p < 1 \text{ and } T > 0$$

subject to 0<p<1 and T>0

Using the principles of calculus the above optimization problem is solved taking partial derivates of $C(T, p)$ with respect to T and equating it to zero, T can be expressed in terms of p as

$$T = g(p) = \frac{1}{bp}\ln\left(\frac{ab(D_2 - D_1)(1-p)}{C_3}\right)$$

where $D_1 = C_1 p + C_1^{'}(1-p)$, $D_2 = C_2 p + C_2^{'}(1-p)$.

Similarly taking partial derivates of $C(T, p)$ with respect to p and equating it to zero, give the numerator as

$$h(p) = (2p-1)C_3(D_2 - D_1)\ln\left\{\frac{ab(D_2 - D_1)(1-p)}{C_3}\right\} - C_1' ab(D_2 - D_1)(1-p)^2 - C_3(C_2' - C_1')(1-p) = 0$$

$h(p)$ is a continuous function of p on (0,1) and $\lim_{p\to 0^+} h(p) = -K$, $\lim_{p\to 1^-} h(p) = -\infty$. where $K = \left(C_3 \ln\left(\frac{ab(C_2' - C_1')}{C_3}\right) + abC_1' + C_3\right)(C_2' - C_1')$.

Differentiating $h(p)$ with respect to p again it can be verified that $h'(p)$ is a continuous and strictly decreasing function on (0,1) where $\lim_{p\to 1^-} h'(p) = -\infty$ and $\lim_{p\to 0^+} h'(p) = 2K + \left(C_3 \ln\left(\frac{ab(C_2' - C_1')}{C_3}\right) + C_1' ab\right)(C_2' - C_2 - C_1' + C_1)$. The optimal release policy can now be determined following Theorem 10.12.

Theorem 10.12 *The optimal values of p and T, denoted by p* and T*, which minimize the expected software cost are determined as*

(a) *If* $K \leq 0$, *then* $p^* = inf\ \{p{:}h(p) < 0\}$ *and* $T^* = g(p^*)$.
(b) *If* $K > 0$, *then define* $p' = \inf\{p : dh/dp < 0\}$ *and*

(i) *If* $h(p') > 0$, *then* $p^* = \min\{C(p_1, T_1), C(p_2, T_2)\}$ *and* $T^* = g(p^*)$, *where* p_1 *and* p_2 *are the solutions to the equations of* $h(p) = 0$ *and* $T_1 = g(p_1)$, $T_2 = g(p_2)$.
(ii) *If* $h(p') = 0$, *then p* equals the unique solution to the equations of* $h(p) = 0$ *and* $T^* = g(p^*)$.
(iii) *If* $h(p') < 0$, *then p* and T* does not exist within* $0 < p < 1$ *and* $T > 0$.

Using the above procedure to find the optimal release time first we need to determine the value $\inf\{p{:}h(p) < 0\}$ or $p' = \inf\{p : dh/dp < 0\}$ whatever is the case assuming a perfect debugging environment, i.e. $p = 1$ as both $h(p)$ and $h'(p)$ functions of p in order to determine the optimal value of p and then using this optimal value of p we estimate the other parameters of the SRGM based on the collected failure data and determine the optimal release time. The procedure if repeated for this optimal value and more dense data, we will obtain another set of optimal values and hence it is an iterative approach. However it is imperative to estimate the level of perfect fault debugging, i.e. p from the SRGM used to describe the failure phenomenon using the collected failure data over a period of time, and not as a decision to be obtained from release time problem by minimizing cost function. The effect of level of perfect debugging on release time can alternatively be obtained by carrying out a sensitivity analysis on the release problem.

Application 10.8

Continuing with Application 10.7, let us first consider that the testing efficiency parameter $p = 1$, then in this case the Kapur and Garg [5] SRGM reduces to the Goel and Okumoto [39] SRGM. So the estimate of the parameters of the SRGM will be same as in Application 10.6, i.e. $a = 142.32$ and $b = 0.1246$. Now let the cost parameters in problem (P13) be $C_1 = \$200$, $C'_1 = \$110$, $C_2 = C'_2 = \$1{,}500$ and $C_3 = \$50$ as in Application 10.7. If the operational mission time $x = 1$ CPU hour, the software life cycle length is $T_l = 2{,}920$ CPU hours, then if testing is stopped after 25 CPU hours of testing, the total cost incurred would be \$55415.25 and the reliability will to the level 0.3654. From Theorem 10.12 the optimal release time minimizing the cost for the software is determined to $T^* = 33.84$ weeks and the optimal level of testing efficiency parameter is $p^* = 0.903$. The optimal cost value is $C(T^*) = \$52168.99$ with the achieved reliability 0.6891. The optimal solution yields a very low level of reliability since there was only cost minimization and no specified requirement of reliability. Hence the policy needs to be improved to include the reliability requirement as well. Graphical plots of cost and reliability growth functions for Applications 10.7 and 10.8 are shown in Figs. 10.15 and 10.16, respectively.

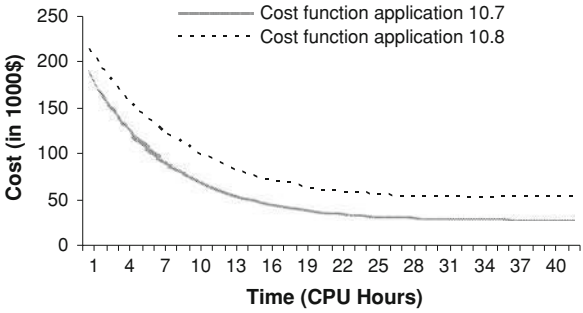


Fig. 10.15 Cost function for Applications 10.7 and 10.8

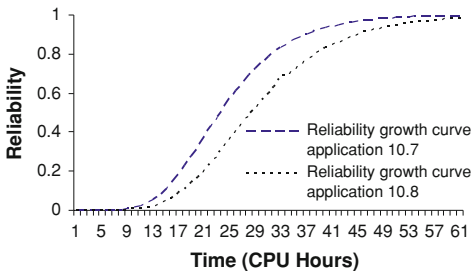


Fig. 10.16 Reliability growth function for Applications 10.7 and 10.8

We can see from the above figures that the cost curve of Application 10.7 lies completely below the cost curve of Application 10.8. Greater value of cost curve in Application 10.8 is due to the fact that testing per unit cost is defined as a function of the testing efficiency parameter p, which increases the total cost value. On the other hand the reliability curve for Application 10.8 lies below the reliability curve of Application 10.7.

10.2.8 Release Policy on Pure Error Generation Fault Complexity Based SRGM

Release policy in the above section is formulated on the pure imperfect fault debugging SRGM. Pham [13] discussed a release policy for a fault complexity based pure fault generation SRGM (refer to Sect. 3.3.4). Along with the traditional cost function they included the penalty cost in the cost function and defined the operational life cycle length to be random. The expected total software development cost is given by

$$C(T) = \sum_{i=1}^{3} C_{i1} m_i(T) + \int_T^{\infty} \left(\sum_{i=1}^{3} C_{i2}(m_i(t) - m_i(T)) \right) g(t) \mathrm{d}t + C_3 T + I(T - T_d) C_p (T - T_d) \tag{10.2.22}$$

where C_{i1}, C_{i2}, i=1,2,3 are the respective costs of critical, major and minor faults in the testing and operational phases, $g(t)$ is the probability density of the life cycle length, $C_p(t)$ is the penalty cost for delay of delivery of the software system and $I(t)$ is an indicator function that is 1 for $t \geq 0$ and 0 otherwise.

The optimal release policy is determined minimizing the cost function based on Theorem 10.13.

$$\text{Minimize} \quad C(T) = \sum_{i=1}^{3} C_{i1} m_i(T) + \int_{T}^{\infty} \left(\sum_{i=1}^{3} C_{i2}(m_i(t) - m_i(T)) \right) g(t) \mathrm{d}t + C_3 T + I(T - T_d) C_p(T - T_d) \tag{P14}$$

Theorem 10.13 *Define* $h(T) = \sum_{i=1}^{3} [C_{i2} R_c(T) - C_{i1}] \lambda_i(T)$, *where* $R_c(T) = \int_T^{\infty} g(t)\mathrm{d}t$, *and* $\lambda_i(T) = m_i'(T)$. *Let* $T_{min} = min\{T_d, T_0\}, T_{max} = min\{T_d, T_0\}, C_{iR} = \frac{C_{i1}}{C_{i2}}$, *Given* C_3, C_{i1}*and* C_{i2}, i = 1, 2, 3. *Assume* $C_{iR} = C_R$, i = 1, 2, 3, *there exist an optimal testing time* T^* for T *that minimizes* $C(T)$ *and the time point is determined based on the following points*.

(a) *If* $h(0) < C3$ *then* $T^* = 0$.
(b) *If* $h(0) \geq C3 > h(Td)$ *then* $T^* = \{T \in [0, T_{min}] : h^{-1}(C_3)\} \equiv T_1$.
(c) *If* $C_3 \leq h(T_d) < C_3 + C_P'(T_d)$ *then* $T^* = Td$.
(d) *If* $C_3 + C_P'(T_d) \leq h(T_d)$ *then* $T^* = \{T \in [T_d, T_{max}] : h(T) - C_p(T) = C_3\} \equiv T_2$.

The release policy (p14) ignores the reliability requirement; the authors have also discussed the release policy under reliability or the remaining number of faults of each type remaining as the constraints. The release policy for minimizing the software cost subject to desired reliability is determined based on the following corollary.

Corollary *Given* C_3, R_0, x, C_{i1} *and* C_{i2} *for* i = 1, 2, 3. *Assume* $C_{iR} = C_R$, i = 1, 2, 3.

(a) *If* $R(x|0) \geq R_0$, *then the optimal policy is the same as in Theorem 10.12*.
(b) *If* $R(x|0) < R_0$ *and*

(i) *If* $h(0) < C_3$*then* $T^* = T_3$.
(ii) *If* $h(0) \geq C_3 > h(T_d)$ *then* $T^* = max\{T_1, T_3\}$.
(iii) *If* $C_3 \leq h(T_d) < C_3 + C_p'(T_d)$ *then* $T^* = \{T_d, T_3\}$
(iv) *If* $C_3 + C_P'(T_d) \leq h(T_d)$ *then* $T^* = \{T_2, T_3\}$

where T_3 is the solution of $\sum_{i=1}^{3} m_i(x) \mathrm{e}^{-(1-\alpha_i) b_i(t)} = \ln\left(\frac{1}{R_0}\right)$,$T_3$ and T_3 are as in Theorem 10.13.

Application 10.9

At an ABC software company a software project on on-line communication systems project was completed in 2000 [48]. The software failure data in the testing phase are collected for 12 weeks and during this testing period a total of 136 failures have been observed. The detected faults are categorized into three categories as critical, major and minor, depending on the severity of the problems. If we choose to Pham [13] fault complexity based SRGM to obtain the measures of testing process (reliability, remaining faults, etc.) then the estimation process yields the parameters of the SRGM given as

$a = 390,\ b_1 = 0.039,\ b_2 = 0.038,\ b_3 = 0.037,\ d_1 = 0.19118,\ d_2 = 0.4046,$ $d_i = 0.4042,\ \alpha_1 = 0.06,\ \alpha_2 = 0.049,\ \alpha_3 = 0.027.$

Now assume that $C_{11} = \$200$, $C_{21} = \$80$ and $C_{31} = \$30$ are the costs of fault removal in the testing phase, $C_{12} = \$1{,}000$, $C_{22} = \$400$ and $C_{32} = \$150$ are the corresponding costs of the operational phase for the critical, major and minor faults, respectively, and C_3 is the testing cost per unit time. Further assume that the software life cycle in the operational phase follows exponential distribution, $g(t)$, with mean life 260 weeks.

$$g(t) = 0.005\,\mathrm{e}^{-t/260}, \quad t > 0$$

If we assume that the scheduled delivery time is $T_d = 25$ weeks and the penalty cost function is

$$C_p(t) = ct = 50t$$

Then following Theorem 10.12 we obtain $T^* = 111.74$ weeks, $C(T^*) = \$36{,}874.98$. For this policy the reliability level of 0.77577 would be reached. Now if we impose a restriction of $R(x|T) \geq 0.8$, then following the corollary in Theorem 10.13 we obtain $T^* = 115.31$ weeks, $C(T^*) = \$40794.26$. Graphical plots of cost and reliability growth functions are shown in Figs. 10.17 and 10.18, respectively.

10.2.9 Release Policy for Integrated Testing Efficiency SRGM

Kapur et al. [47] proposed an SRGM integrating the effect of both imperfect fault debugging and error generation (refer Sect. 3.5.2). They discussed that the increase in fault content of software due to fault generation has a direct effect on the software cost similar to the effect due to imperfect debugging. Since the testing cost parameter C_3 depends on the testing team composition and testing strategy used, if the probability of perfect debugging is to be increased and probability of error generation is to be decreased, it is expected that extra financial resources will be needed to engage more experienced testing personnel, and this will result in an increase of C_3. In other words, C_3 should be a function of both the testing level and error generation, denoted by $C_3(p,\alpha)$ and hence this function should possess the following two properties:

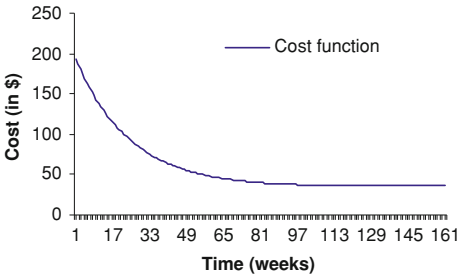


Fig. 10.17 Cost function for Application 10.9

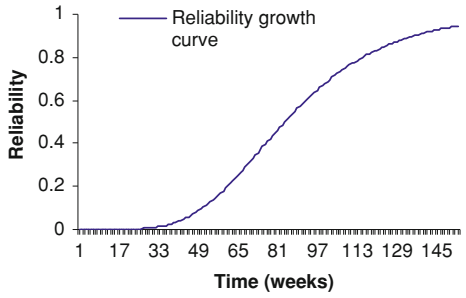


Fig. 10.18 Reliability growth function for Application 10.9

1. $C_3(p,\alpha)$ is a monotonous increasing function of p and $(1-\alpha)$
2. When $p \to 1$ and $\alpha \to 0$, $C_3(p, \alpha) \to \infty$.

The second property implies that perfect debugging is impossible in practice or the cost of achieving it is extremely high. A simple function that meets the above two properties above is given by

$$C_3(p,\alpha) = \frac{C_3}{(1-p(1-\alpha))}$$

The optimization problem minimizing the total expected software cost in order to determine optimal release time T^* subject to the software reliability not less than a specified reliability objective can be formulated as follows

$$\begin{aligned}\text{Minimize} \quad & C(T,p,\alpha) = \big(C_1 p + C_1'(1-p)\big)m_f(T) \\ & + \big(C_2 p + C_2'(1-p)\big)\big(m_f(T_l) - m_f(T)\big) + \frac{C_3 T}{(1-p(1-\alpha))} \\ \text{Subject to} \quad & R(x|T) = \exp[-(m(T+x) - m(T))] \geq R_0 \\ & \text{where } 0 < R_0 < 1 \quad \text{and} > 0. \end{aligned} \tag{P15}$$

To determine the optimal release time taking partial derivative of $C(T)$ with respect to T and equating it to zero we obtain

$$\lambda(t) = \frac{C}{(D_2 - D_1)(1 - p(1-\alpha))} \tag{10.2.23}$$

where D_1, D_2 as defined in Theorem 10.11, $\lambda(t) = ab \exp(-bp(1-\alpha)t)$, $\lambda(0) = ab$ and $\lambda(\infty) = 0$, $\lambda(t)$ is a decreasing function in time. If $ab > \frac{C}{(D_2-D_1)(1-p(1-\alpha))}$ then $C(T)$ is decreasing for $T < T_0$ and increasing for $T > T_0$ thus, there exists a finite and unique $T = T_0(>0)$ minimizing the total expected cost. And if $ab \leq \frac{C}{(D_2-D_1)(1-p(1-\alpha))}$ then $C'(T) > 0$ for $T > 0$ and hence $C(T)$ is minimum for $T = 0$. Also

$$R(x|0) = \mathrm{e}^{-m(x)}, \quad R(x|\infty) = 1 \tag{10.2.24}$$

It is known that $R(x|t), t > 0$ is an increasing function of time. Thus $R(x|0) < R_0$ there exists $T = T_1(>0)$ such that $R(x|T) = R_0$ and if $R(x|0) \geq R_0$ then $R(x|t) \geq R_0 \quad \forall \quad t \geq 0$ and $T = T_1 = 0$. Combining the cost and reliability requirements the following theorem determines the optimal release policy.

Theorem 10.14 *Assuming* $C_2 > C_1 > 0$, $C_2' > C_1' > 0$, $x > 0$ *and* $0 < R_0 \leq 1$

(a) *If* $ab > \frac{C_3}{(D_2-D_1)(1-p(1-\alpha))}$ *and* $R(x|0) < R_0 < 1$ *then* $T^* = max\ (T_0, T_1)$
(b) *If* $ab > \frac{C_3}{(D_2-D_1)(1-p(1-\alpha))}$ *and* $R(x|0) \geq R_0 > 0$*then* $T^* = T_0$
(c) *If* $ab \leq \frac{C_3}{(D_2-D_1)(1-p(1-\alpha))}$*and* $R(x|0) < R_0 < 1$ *then* $T^* = T_1$
(d) *If* $ab \leq \frac{C_3}{(D_2-D_1)(1-p(1-\alpha))}$ *and* $0 < R_0 \leq R(x|0)$ *then* $T^* = 0$

Application 10.10

The parameters *a, b, p* and α of the SRGM are estimated again using the data set from Application 10.6 and the estimated values of the unknown parameters be $a = 134$, $b = 0.14024$, $p = 0.99842$ and $\alpha = 0.01256$. Let the cost parameters are $C_1 = \$200$, $C_1' = \$110$, $C_2 = C_2' = \$1{,}500$ and $C_3 = \$10$. If minimum reliability requirement by the release time is 0.85, then following Theorem 10.14 we obtain $T^* = 31.57$ CPU hours. The minimum total expected software cost at T^*, i.e. $C(T^*) = \$138641.52$. Graphical plots of cost and reliability growth functions are shown in Figs. 10.19 and 10.20, respectively.

Authors also showed a sensitive analysis on the optimal release policy to study the effect of variations in minimum reliability requirement by the release time, most sensitive costs involved in cost function and level of perfect debugging, on the optimal release time and total expected software testing cost.

Define

$$\text{Relative change (RC)} = \frac{\text{MOV} - \text{OOV}}{\text{OOV}} \tag{10.2.25}$$

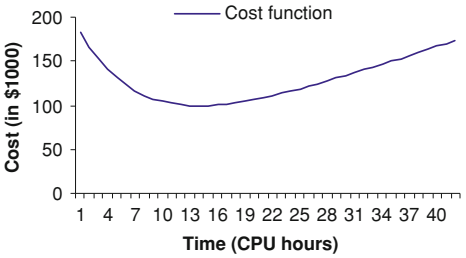


Fig. 10.19 Cost function for Application 10.10

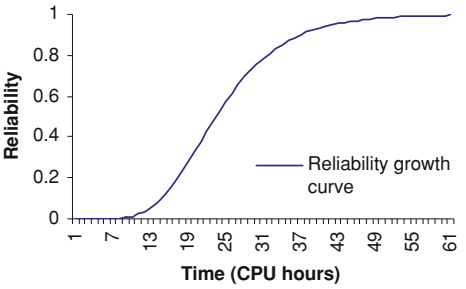


Fig. 10.20 Reliability growth function for Application 10.10

where OOV is the original optimal values and MOV is the modified optimal values obtained when there is a variation is some attribute of the release time problem.

10.2.9.1 Effect of Variations in Minimum Reliability Requirement by the Release Time

The optimal value of the release time obtained for the desired reliability level may be too late as compared to the scheduled delivery time, in such a case the management and/or the user of a project-based software may agree to release the software at some lower reliability level with some warranty on the failures, which in turn will change the optimal release time to an earlier time and consequently lower the cost. On the other hand if the scheduled delivery is later than the optimal release time the management may wish to increase the desired reliability level at some additional testing cost.

Assuming the values of parameters and various costs associated with cost model to be same as above. If minimum reliability requirement by the release time increased to 0.95 (about 12% increase) then we obtain $T^* = 42.21$ CPU hours (about 33.7% increase) and its RC is 0.33703. The minimum total expected software cost at T^*, i.e. $C(T^*) = \$174587.11$ (about 25.93% increase), its RC is 0.25927 and if minimum reliability requirement by the release time decreased to 0.75 (about 12% decrease) then we obtain $T^* = 29.73$ CPU hours (about 5.83% decrease) and its RC is -0.05828. The minimum total expected software testing cost at T^*, i.e. $C(T^*) = \$132776.98$ (about 4.23% decrease), its RC is -0.0423.

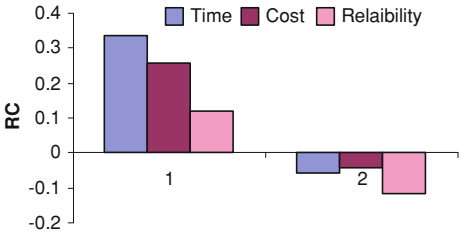


Fig. 10.21 Relative change in release policy for 12% increase and decrease in reliability

Figure 10.21 plots the relative change in the optimal release time and cost for the case of 12% increase and decrease in reliability objective.

10.2.9.2 Effect of Variations in Level of Perfect Fault Debugging

Now investigate the sensitivity of variations in level of perfect fault debugging parameter p. If the testing personals were skilled personal the level of perfect fault debugging would be more or vice versa. Variations in level of perfect debugging have significant effect on the optimal time of software release. If the level of perfect debugging increases for a testing process it is expected that the software can be released earlier as compared to the optimal release time determined otherwise and vice versa.

Assume the values of parameters a, b and α of the SRGM and the cost involved in cost function to be same as above with reliability requirement 0.85. Let the testing efficiency parameter $p = 0.9$, then we have $T^* = 37.63$ CPU hours and $C(T^*) = \$42480.16$. Now if p is to be increased by 10%, i.e. $p = 0.99$, then we obtain $T^* = 43.16$ CPU hours (about 9.22% decrease) and its RC is -0.09221 and with $C(T^*) = \$102307.67$ (about 140.83% increase), its RC is 1.40836. On the other hand if p decreases by 10%, i.e. $p = 0.81$ then we obtain $T^* = 41.86$ (about 11.24% increase) and its RC is 0.11241. The minimum total expected software cost at T^* i.e. $C(T^*) = \$35601.43$ (about 16.19% decrease), its RC is -0.16193. Figure 10.22 plots the relative change in the optimal release time and cost for the case of 10% increase and decrease in perfect fault debugging parameter p.

Similar conclusion can be obtained for the costs and the other parameters of the SRGM such as $C_1, C_1', C_2, C_2', C_3, a, b$ and α. Such an analysis helps to determine the changes in release time, cost, reliability, etc. when some parameter of the optimization problem changes during the testing process without resolving the problem as a new.

10.2.10 Multi-Criteria Release Time Problems

The release time problems discussed in this chapter are all considering single objective of either cost minimization or reliability maximization. Some of the problems are unconstrained while others have the lower and/or upper bound type

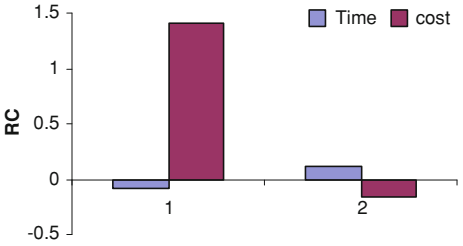


Fig. 10.22 Relative change in release policy for 10% increase and decrease in perfect fault debugging parameter p

constraints. Unconstrained optimization of cost or reliability most often provides solution which is not acceptable to the developer or user or both as seen in Application 10.1. It encouraged the researchers to formulate constrained problems with lower bound on reliability when the objective is cost minimization and vice versa. A lower bound of 0.85 on reliability may result in substantial increase in cost as compared to unconstrained cost minimization or reliability maximization under a given budget may yield a solution with very low level of reliability. In practice cost minimization and reliability maximization are the conflicting objectives for determining the release time and requires tradeoff between them. Kapur et al. [11] started multi-objective optimization in release time determination. They propose to make reliability maximization and cost minimization as two simultaneous objectives of the release time problem and then assigning weights to the two objectives according to their relative importance one can find the optimal solution. Such a problem is specifically called a bi-criterion release time optimization problem. We can also impose bounds on one or both of the objectives of the problem. The problem considered by the authors considers minimization of the total expected software cost and maximization of reliability simultaneously such that total expected cost during the software life cycle does not exceed the specific budget and conditional reliability is not less than a prescribed reliability objective. The solution procedure is discussed for exponential and S-shaped SRGM.

$$\begin{aligned} &\text{Maximize} && R(x|T) \text{ (or } \log R(x|T) \\ &\text{Minimize} && C(T) \text{ (or } \bar{C}(T)) \\ &\text{Subject to} && C(T) \le C_B \text{ (or } \bar{C}(T) \le 1) \\ & && R(x|T) \ge R_0 \\ & && T \ge 0,\ 0 < R_0 < 1 \end{aligned} \tag{P15}$$

where $\bar{C}_T = C_T/C_B; \log R(x|T) = m(T) - m(T+x); \bar{C}(T) = \bar{C}_1 m(T) + \bar{C}_2(m(T_l) - m(T)) + \bar{C}_3 T; \bar{C}_i = C_i/C_B$, $i = 1, 2, 3$ and $m(T)$ is the mean value function of the SRGM.

In this formulation we either maximize reliability or log of reliability as we know that maximization of a function is same as maximization of its log function. This is the usual procedure to state the problem with cost minimization objective but the problem is normalized before solving to bring both the objectives on the same scale, i.e. having their value lie in the range of (0,1), i.e. minimization of

$\bar{C}(T)(\bar{C}_T = C_T/C_B)$. It may be noted that the values 0 and 1 can be included in or excluded from the set, it depends on the objective. For example in case of cost minimization we may spend the whole budget so 1 is included while reliability level of 1 practically impossible hence it is excluded in this case.

The approach of multi-criteria optimization suggests to reduce the problem to a single objective by introducing $\lambda = |\lambda_i, i = 1, \ldots, n| \in R^n, \lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i = 1$, where λ_i $(i = 1, 2)$ is the weight assigned to the ith objective, and n is the total number of objectives. Using λ_1 and λ_2 (P15) is reformulated as

$$\begin{aligned} \text{Maximize} \quad & F(T) = \lambda_1 \log\ R(x|T) - \lambda_2 \bar{C}(T) \\ \text{Subject to} \quad & R(x|T) \geq R_0 \\ & \bar{C}(T) \leq 1 \\ & T \geq 0, 0 < R_0 < 1. \end{aligned} \tag{P16}$$

Such a release policy gives enough flexibility to the software developers to find out the optimal release time based on their priority in respect of reliability and cost components. If reliability is more important then higher weight may be attached to reliability objective as in case of safety critical projects. Similarly, for business application software packages, more weight may be attached to the cost objective. This form introduces flexibility over the earlier release policies where we optimized either the cost or reliability functions.

Kapur et al. [11] suggested to use the method of calculus to solve the above problem, similar to the approach followed in case of the single optimization of release time discussed throughout the chapter. The theorem given to solve the problem is very long and complex and requires a lot of time to determine the solution. Throughout the book we have favored the use of software for the purpose of computation. A number of software are available which can be used to solve large size optimization problems with very little effort and in very less time such as LINGO, LINDO, QSB, MATLAB, etc. For detailed analytical solution reader can refer to the original manuscript Kapur et al. [11]. Here we use the software package LINGO to solve the problem for different values of weights for the two objectives.

Application 10.11

We consider the problem based on Goel and Okumoto [39] exponential SRGM and the data set taken in Application 10.1 with the same specification of cost parameters and the estimated values of the unknown parameters, i.e. $a = 130.30$, $b = 0.083$, $C_1 = \$10$, $C_2 = \$50$ and $C_3 = \$100$ and software life cycle length $T_l = 156$ weeks. Let operational reliability requirement $R_0 = 0.75$ for $x = 1$ and budget $C_B = 20{,}000$. Using these information we solved the problem for the different values of the weights λ_1 and λ_2. The result is summarized in Table 10.3.

It can be seen from the above table that when more weight is attached to the reliability objective release of the software gets delayed with added cost and vice

Table 10.3 Summary of bi-criteria release policy for different weights of the objectives

λ_1	λ_2	T^* (in weeks)	$\bar{C}(T^*)$	C (T^*) (in \$)	R (T^*)
0.9	0.1	88.54	0.5080	10160.74	0.9933
0.8	0.2	78.81	0.4596	9192.07	0.9851
0.7	0.3	72.37	0.4277	8553.34	0.9748
0.6	0.4	67.12	0.4017	8035.13	0.9613
0.5	0.5	62.34	0.3783	7566.17	0.9429
0.4	0.6	57.60	0.3553	7106.54	0.9166
0.3	0.7	52.51	0.3311	6621.22	0.8756
0.2	0.8	46.49	0.3031	6061.82	0.8033
0.1	0.9	43.20	0.2884	5767.51	0.7500

versa. The practitioners can solve the problem for the different values of the weights and accept the most desirable solution.

10.2.11 Release Problem with Change Point SRGM

In Chap. 5 we have discussed several change point SRGM. As we have already discussed in the chapter that these SRGM often provide better fit then the models which does not consider the changing behavior of the testing process. Owing to the importance of change point models in the reliability estimation Kapur et al. [49] formulated release policy for the exponential change point SRGM. The simple cost model (10.2.1) is modified to include separate cost of fault removal before and after change point.

$$C(T) = C_1 m_1(\tau) + C_1'(m_2(T) - m_1(\tau)) + C_2(a - m_2(T)) + C_3 T \qquad (10.2.26)$$

In the above cost model C_1 is the fault removal cost per fault before the change point τ, in the testing phase and C_1' is the corresponding cost after change point and before release. Other costs are same as in (10.2.1). Here the third component of cost function is changed to $C_2(a - m_2(T))$ in contrast to $C_2(m_2(T_l) - m_2(T))$. The component has entirely the same behavior as $m_2(T_l) \simeq a$. Here $m_1(t)$ and $m_2(t)$ are mean value functions for the fault removal process before and after the change point (refer to Sect. 5.4.1). The release policy can be stated as

$$\begin{aligned} \text{Minimize} \quad & C(T) = C_1 a\left(1 - \mathrm{e}^{-b_1\tau}\right) + C_1' a\left(\mathrm{e}^{-b_1\tau} - \mathrm{e}^{-b_1\tau - b_2(T-\tau)}\right) \\ & + C_2 a \mathrm{e}^{-b_1\tau - b_2(T-\tau)} + C_3 T \\ \text{Subject to} \quad & R(x|T) \geq R_0 \end{aligned} \qquad \text{(P17)}$$

To find the optimal solution of the problem the cost function is differentiated with respect to T and equated to zero, i.e.

$$-ab_2\left(C_2 - C_1'\right)\mathrm{e}^{-b_1\tau - b_2(T-\tau)} + C_3 = 0$$

Here if $ab_2(C_2' - C_1') \le C_3$ then the cost function will be monotonically increasing and will be minimum for $T = \tau$. If $ab_2(C_2' - C_1') > C_3$ then there exists a finite T (say T_0) such that $C'(T) = 0$. In this case the cost function $C(T)$ first decreases for $T < T_0$ then increases for $T > T_0$. In this case $C(T, \tau)$ will be minimum for $T = T_0$, where $T_0 = \frac{1}{b_2}\ln\left(\left(ab_2(C_2' - C_1')e^{(b_2-b_1)\tau}\right)/C_3\right)$. Now $R(x|0) = e^{-m(x)}$; $R(x|\infty) = 1$ and $R(x|T)$ is an increasing function for $T > 0$. Differentiating $R(x|T)$ with respect to T, we get

$$R'(x|T) = a\left[b_2 e^{-b_1\tau - b_2(T-\tau)} - b_2 e^{-b_1\tau - b_2(T+x-\tau)}\right]$$

$R'(x|T) > 0$ for $\forall \quad T > 0$ thus, if $R(x|0) = R_0$. there exists $T = T_1(>0)$ such that $R(x|T_1) = R_0$. The optimal release policy can be obtained from Theorem 10.15.

Theorem 10.15 *Given that* $C_1 < C_1' < C_2$

(a) *If* $ab_2(C_2' - C_1') \le C_3$ *and* $R(x|\tau) \ge R_0$, *then* $T^* = \tau$
(b) *If* $ab_2(C_2' - C_1') \le C_3$ *and*, $R(x|\tau) < R_0$ *then* $T^* = T_1$
(c) *If* $ab_2(C_2' - C_1') > C_3$ *and* $R(x|\tau) \ge R_0$,*then* $T^* = T_0$
(d) *If* $ab_2(C_2 - C_1') > C_3$ *and* $R(x|\tau) < R_0$, *then* $T^* = max\ (T_0, T_1)$

Application 10.12

In Application 10.1 we have obtained the release policy minimizing cost subject to the required reliability level based on exponential SRGM without change point [39]. We use the same failure data and cost parameter to estimate the parameters of the change point exponential SRGM and obtain the release policy in this application. The estimated values of the parameters of the SRGM for the change point equal to 8 weeks, i.e. $\tau = 8$ weeks, are $a = 103.599$, $b_1 = 0.105$ and $b_2 = 0.24$. The cost parameters are $C_1 = \$5$, C_1'=\$10, $C_2 = \$50$ and $C_3 = \$500$. The operation reliability requirement is $R_0 = 0.80$ for mission time $x = 1$. Applying Theorem 10.15 we obtain $T^* = 23.65$ weeks and $C(T^*) = \$12608.44$. The cost function is of ever increasing type. The cost and reliability curves are shown in Figs. 10.23 and 10.24.

If we decrease the testing cost C_3 from \$500 to \$100 then the cost function first decreases and then increases. The cost function for this case is shown in Fig. 10.25. It attains its minima at the 14.07 week time, i.e. $T_0 = 14.07$ weeks.

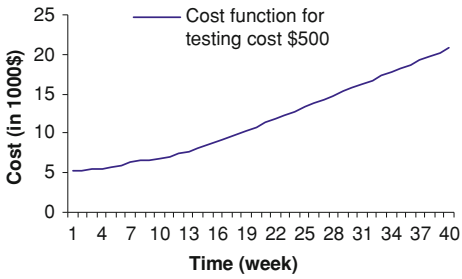


Fig. 10.23 Cost function for the Application 10.12 ($C_3 = \$500$)

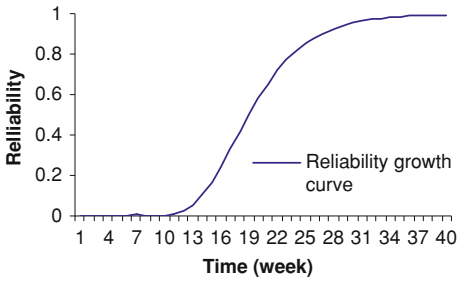


Fig. 10.24 Reliability growth curve for Application 10.12

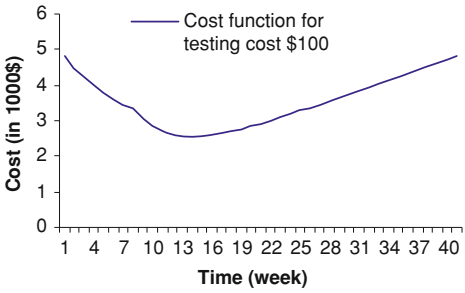


Fig. 10.25 Cost function for the Application 10.12 ($C_3 = \$100$)

The optimal release policy for this case corresponds to part (d) of Theorem 10.15 and is given as $T^* = \max(T_0 = 14.07, T_1 = 23.65) = 23.65$ weeks and $C(T^*) = \$3148.44$. One can compare the results obtained from this policy with the one for SRGM without change point. Policy (P17) suggests an early release.

10.3 Fuzzy Optimization in Software Release Time Decision

We have discussed several release policies under crisp environment in the previous section. The Sect. 10.1 describes the limitations of using crisp optimization in release time and the role of fuzzy set theory and fuzzy optimization in release time optimization. In this section we are describing in detail how to formulate a fuzzy optimization problem for the release time determination and the solution approach with numerical application. The release policy is formulated with cost minimization goal subject to the failure intensity constraint defined under fuzzy environment.

10.3.1 Problem Formulation

Gupta [22] formulated a release time optimization problem with cost minimization objective based on the SRGM for error removal phenomenon due to Kapur et al. [50]. Author claimed that this model is chosen for various reasons such that the model is simple, easy to apply, all the parameters of the model have clear

interpretation, not too many unknown parameters are involved and is a flexible model in the sense that it can describe either an exponential or S-shaped failure curve depending on the values of the parameters (refer to Sect. 2.3.7).

10.3.1.1 The Cost Model

Section (10.2.5) describes the release policy for the Kapur and Garg [7] SRGM. The cost model (10.2.12) is first modified to include the risk cost of field failure. The risk is measured by the unreliability of the software, hence including this cost in the cost model takes care of the reliability objective and hence the quality from the users, point of view. Incorporating the failure intensity constraint on the other hand specifies the quality from the point of view of the developer. Hence in this way quality level is satisfied both from the users, as well as developers point of view.

Introducing the risk cost of failure in field and rearranging the like terms the modified cost model (10.2.12) is given as

$$\begin{aligned} C(T) = {} & C_1 m_f(T) + C_1'\big(m_r(T) - m_f(T)\big) + \big(C_2 - C_2'\big)\big(m_f(T_l) - m_f(T)\big) \\ & + C_2'(m_r(T_l) - m_r(T)) + C_3 T + C_4(1 - R(x|T)) \end{aligned} \tag{10.3.1}$$

The values of cost function constant coefficients $C_i, i = 1, 2, 3, 4$ and $C_i', i = 1, 2$ depend on a number of factors such as testing strategy, testing environment, team constitution, skill and efficiency of testing and debugging teams, infrastructure, etc., which are non-static and are subject to change during testing. The information and data available to compute these quantities are usually defined imprecisely. Defining a fuzzy model of the above cost function provides us a method to deal with these uncertainties directly. The cost function (3.1.8) can be defined under fuzzy environment as

$$\begin{aligned} \tilde{C}(T) = {} & \tilde{C}_1 m_f(T) + \tilde{C}_1'\big(m_r(T) - m_f(T)\big) + \big(\tilde{C}_2 - \tilde{C}_2'\big)\big(m_f(T_l) - m_f(T)\big) \\ & + \tilde{C}_2'(m_r(T_l) - m_r(T)) + \tilde{C}_3 T + \tilde{C}_4(1 - R(x|T)) \end{aligned} \tag{10.3.2}$$

The cost coefficients represents that they are fuzzy numbers. Fuzzy numbers are assumed to be Triangular Fuzzy Numbers (TFN) [51].

The problem is formulated with cost minimization objective with a lower bound on the desired quality level in terms of failure intensity to be achieved by the release time. In most of the cases developers provide ambiguous statements on the bounds as they want to be flexible due to competitive considerations and a slight shift on bounds can provide more efficient solutions. It renders the resource and requirement constants of the problem vague and soft inequalities in the constraints. Fuzzy cost function, soft inequalities and ambiguous statements by the developers make it necessary to define the SRTD problem under fuzzy environment. The problem considered here can now be stated as

$$\begin{aligned} &\text{Minimize} \quad \tilde{C}(T) \\ &\text{Subject to} \quad \lambda(T) \lesssim \tilde{\lambda}_0 \\ &\qquad T \geq 0 \end{aligned} \tag{P18}$$

The symbol $\gtrsim (\lesssim)$ is called "fuzzy greater (less) than or equal to" and has linguistic interpretation "essentially greater (less) than or equal to". Crisp optimization techniques cannot be applied directly to solve the problem since these methods provide no well-defined mechanism to handle the uncertainties quantitatively. Fuzzy optimization approach is used here to solve the problem. The problem with cost minimization objective subject to achieving a desired level of failure intensity can also be considered as a multiple objective problem of cost and failure intensity minimization while solving with the fuzzy optimization. The two objectives can be assigned different weights according to the relative importance and the problem can be solved with the fuzzy weighted min–max approach.

10.3.2 Problem Solution

Algorithm 10.1 specifies the sequential steps to solve the fuzzy mathematical programming problems. Figure 10.26 illustrates the solution methodology in the form of a flowchart.

Algorithm 10.1

Step 1: compute the crisp equivalent of the fuzzy parameters using a defuzzification function (ranking of fuzzy numbers). Same defuzzification function is to be used for each of the parameters. We use the defuzzification function of type $F_2(A) = (a_1 + 2a + a_u)/4$.

Step 2: incorporate the objective function of the fuzzifier min (max) as a fuzzy constraint with a restriction (aspiration) level. The inequalities are defined softly if the requirement (resource) constants are defined imprecisely.

Step 3: define appropriate membership functions for each fuzzy inequality as well as constraint corresponding to the objective function. The membership functions for the fuzzy numbers less than or equal to and greater than or equal to type are given as

$$\mu(T) = \begin{Bmatrix} 1 & ; G(T)_0 \\ \frac{G^* - G(T)}{G^* - G_0} & ; G_0 < G(T) \leq G^* \\ 0 & ; G(T) > G^* \end{Bmatrix}$$

$$\widehat{\mu}(T) = \begin{Bmatrix} 1 & ; Q(T) \geq Q_0 \\ \dfrac{Q(T) - Q^*}{Q_0 - Q^*} & ; Q^* \leq Q(T) < Q_0 \\ 0 & ; Q(T) < Q^* \end{Bmatrix}$$

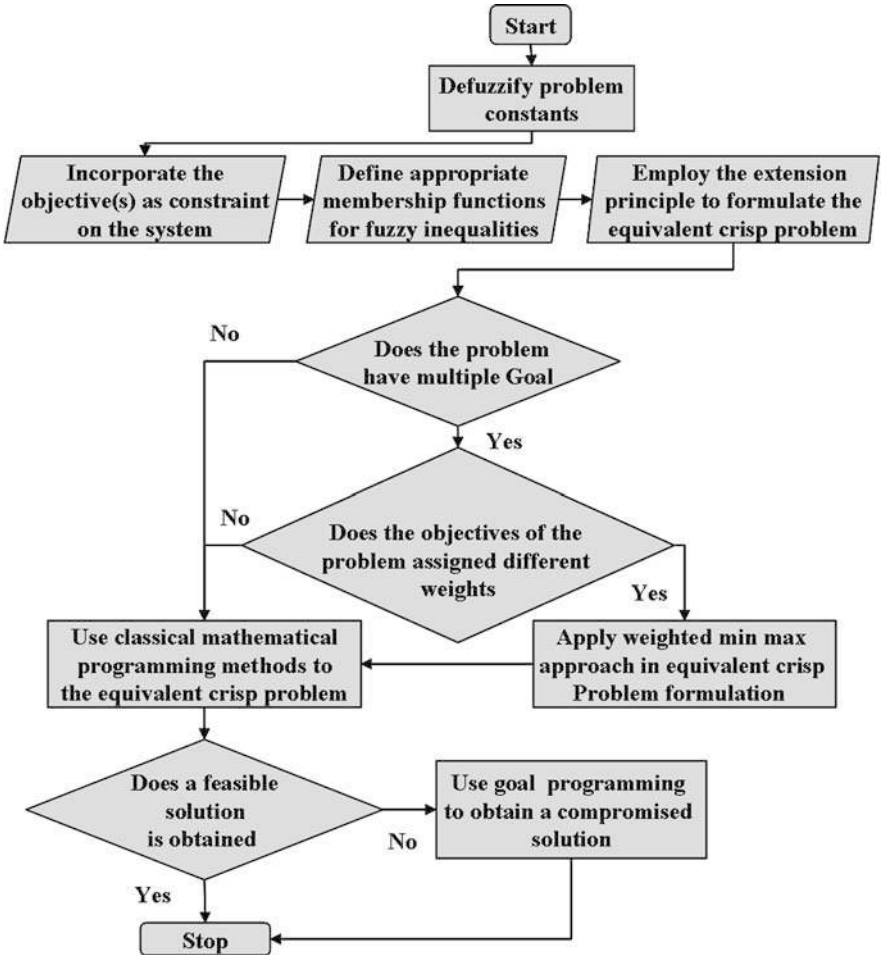


Fig. 10.26 A flowchart of solution procedure for fuzzy optimization problem

respectively, where G_0 and Q_0 are the restriction and aspiration levels, respectively, and G^* and Q^* are the corresponding tolerance levels. The membership functions can be a linear or piecewise linear function that is concave or quasiconcave.

Step 4: Employ extension principle [51] to identify the fuzzy decision, which results in a crisp mathematical programming problem given by

$$\begin{aligned} &\text{Maximize} \quad \alpha \\ &\text{Subject to} \quad \mu_i(T) \geq \alpha, i = 1, 2, \ldots n; \\ &\alpha \geq 0, \alpha \leq 1, T \geq 0 \end{aligned} \tag{P$*$}$$

(P^*) can be solved by the standard crisp mathematical programming algorithms.

Step 5: While solving the problem following steps 1–4, objective of the problem is also treated as a constraint. In the release time decision problem under consideration each constraint corresponds to one major factor effecting the release time. Hence we can consider each constraint to be an objective for the decision maker and the problem can be looked as a fuzzy multiple objective mathematical programming problem. Further each objective can have different level of importance and can be assigned weight to measure the relative importance. The resulting problem can be solved by the fuzzy weighted *min–max* approach. The crisp formulation of the weighted problem is given as

$$\begin{aligned} &\text{Maximize} && \alpha \\ &\text{Subject to} && u_i(T) = w_i\alpha, \quad i = 1, 2, \ldots, n \\ &\alpha \geq 0, && \alpha \leq 1,\ T \geq 0, \\ & && \sum_{i=1}^{n} w_i = 1 \end{aligned} \tag{P**}$$

where n is the number of constraints in (P^{**}) and α represents the degree up to, which the aspiration of the decision maker is met. The problem (P^{**}) can be solved using standard mathematical programming approach.

Step 6: If a feasible solution is not obtainable for the problem (P^{*}) or (P^{**}) then we can use fuzzy goal programming approach to obtain a compromised solution [32]. The method is discussed in detail in the numerical example.

In the next section using the above algorithm we give an application of the above algorithm.

Application 10.13

For this application we again consider the data set from Application 10.6. Estimated values of parameters for the Kapur and Garg [7] SRGM are $a = 147$, $p = 0.11073$ and $q = 0.012$. The fuzzy cost coefficient constants $\tilde{C}_i, i = 1, 2, 3, 4$ and $\tilde{C}'_i, i = 1, 2$ and the minimum level of failure intensity desired at the release time $\tilde{\lambda}_0$ are specified as TFN represented as $A = (a_1, a, a_u)$. The values of these fuzzy numbers are specified by the management based on the past experience and/or expert opinion. We choose the defuzzification function $F_2(A) = (a_1 + 2a + a_u)/4$ to defuzzify the fuzzy numbers. The TFN corresponding to the cost coefficients and failure intensity aspiration are tabulated in Table 10.4. Defuzzified values of these parameters are also given in the table.

The Problem (P18) is restated using the defuzzification function $F_2(A)$ as

$$\begin{aligned} &\text{Minimize} && F_2(\tilde{C}(T)) \\ &\text{Subject to} && \lambda(T) \lesssim F(\tilde{\lambda}_0) \\ & && T \geq 0 \end{aligned} \tag{P19}$$

Table 10.4 Triangle fuzzy and defuzzified values of the cost coefficients (in $) and intensity aspiration level

Fuzzy parameter (A)	a_1	a	a_u	Defuzzified value ($F_2(A)$)
C_1	4	4.6	4.8	4.5
$\tilde{C}_1'$	8	10	12	10
C_2	4	5.2	5.6	5
$\tilde{C}_2'$	22	25.5	27	25
C_3	18	20.5	21	20
C_4	2,700	2,900	3,500	3,000
λ	0.001	0.0011	0.0016	0.0012

where

$$\begin{aligned} F_2(\tilde{C}(T)) = {} & F_2(\tilde{C}_1)T + F_2(\tilde{C}_2)m_f(T) \\ & + \left(F_2(\tilde{C}_4) - F_2(\tilde{C}_5)\right)\left(m_f(T_{lc}) - m_f(T)\right) + F_2(\tilde{C}_3)\left(m_r(T) - m_f(T)\right) \\ & + F_2(\tilde{C}_5)(m_r(T_{lc}) - m_r(T)) + F_2(\tilde{C}_6)(1 - R(x|T)) \end{aligned}$$

Assume the software life cycle in operational phase to be 6 months $= T_{lc} = T + 4{,}032$ h (release time + number of hours in 6 months). Using the values of TFN given in Table 10.4 and substituting in the defuzzification function $F_2(A)$ to obtain the defuzzified values of these constant coefficients, the Problem (P19) is rewritten as

$$\begin{aligned} \text{Minimize} \quad & C(T) = 4.5T + 10m_f(T) + 5\left(m_r(T) - m_f(T)\right) \\ & \quad + 5\left(m_f(T + 4,032) - m_f(T)\right) + 20(m_r(T + 4,032) - m_r(T)) \\ & \quad + 3,000(1 - R(4,032|T)) \\ \text{Subject to} \quad & \lambda(T) \lesssim 0.0012 \\ & T \geq 0 \end{aligned} \tag{P20}$$

where $m(T) = \dfrac{147 * 0.11073}{0.0012} \ln\left[\dfrac{0.11073 + 0.0012}{0.11073 + 0.0012e^{-(0.11073+0.0012)T}}\right]$

$$m(T + 4032) = \frac{147 * 0.11073}{0.0012} \ln\left[\frac{0.11073 + 0.0012}{0.11073 + 0.0012e^{-(0.11073+0.0012)(T+4032)}}\right]$$

and $\lambda(t) = \dfrac{147 * (0.11073 + 0.0012)e^{-(0.11073+0.0012)T}}{1+(0.0012/0.11073)e^{-(0.11073+0.0012)T}}$.

Now the cost objective function is introduced as a constraint with imprecise definition of the available budget. If the available budget is specified as a TFN given as $\tilde{C}_0$ =($1,815, $1,855, $1,875) again using the defuzzification function $F_2(A)$ we get $F_2(\tilde{C}_0) = C_0 =$ $1,850. The Problem (P20) can now be restated as

$$\begin{aligned}&\text{Find } T\\&\text{Subject to } 4.5T+10m_f(T)+5\big(m_r(T)-m_f(T)\big)+5\big(m_f(T+4{,}032)-m_f(T)\big)\\&\qquad+20(m_r(T+4{,}032)-m_r(T))+3{,}000(1-R(4{,}032|T))\lesssim 1850\\&\qquad\lambda(T)\lesssim 0.0012\\&\qquad T\geq 0\end{aligned}\tag{P21}$$

The membership functions $\mu_i(T)$; $i = 1, 2$ for each of the fuzzy inequalities in Problem (P21) are defined. Definition of membership function requires upper tolerance level in the cost (C^*) and failure intensity (λ^*). Let $C^* = \$1{,}950$ and $\lambda^* = 0.0015$, then

$$\mu_1(T)=\begin{cases}1 & C(T)<1{,}850\\ \dfrac{1{,}950-C(T)}{19{,}050-1{,}850} & 1{,}850\leq C(T)\leq 1{,}950\\ 0 & C(T)>1{,}950\end{cases}\tag{10.3.3}$$

$$\mu_2(T)=\begin{cases}1 & \lambda(T)<0.0012\\ \dfrac{0.0015-\lambda(T)}{0.0015-0.0012} & 0.0012\leq\lambda(T)\leq 0.0015\\ 0 & \lambda(T)>0.0015\end{cases}\tag{10.3.4}$$

The cost and failure intensity curves plotted on time scale are shown in Figs. 10.27 and 10.28. The cost and failure intensity membership functions plotted on cost and failure intensity scales, respectively, are shown in Figs. 10.29 and 10.30.

Here it can be seen that the membership functions are piecewise linear (quasiconcave). Now we formulate the crisp optimization problem to identify the fuzzy decision based on extension principle and solve the fuzzy system of inequalities corresponding to the problem.

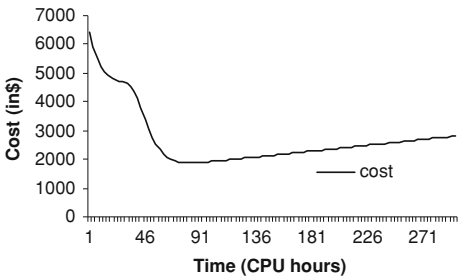


Fig. 10.27 Cost curve for Application 10.13

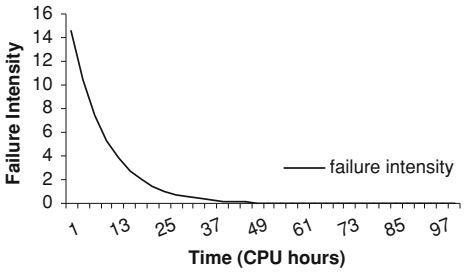


Fig. 10.28 Failure intensity curve for Application 10.13

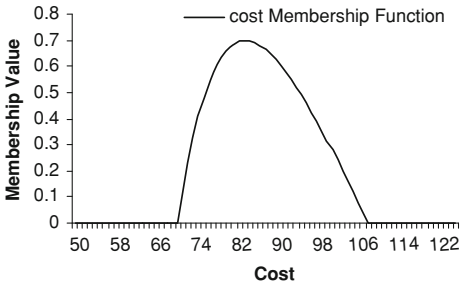


Fig. 10.29 Cost membership function for Application 10.13

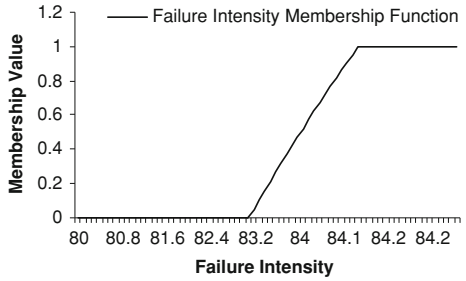


Fig. 10.30 Failure intensity function for Application 10.13

$$
\begin{aligned}
&\text{Maximize} \quad \alpha \\
&\text{Subjectto} \quad \mu_1(T) = \frac{1950 - C(T)}{1950 - 1850} \geq \alpha \\
&\qquad\qquad \mu_2(T) = \frac{0.0015 - \lambda(T)}{0.0015 - 0.0012} \geq \alpha \qquad \text{(P22)} \\
&\qquad\qquad a \geq 0 \\
&\qquad\qquad \alpha \geq 0 \ \alpha \leq 1 \\
&\qquad\qquad T > 0
\end{aligned}
$$

The Problem (P22) is a crisp non-linear programming problem and can be solved using standard mathematical programming methods. Mathematical software such as LINGO, LINDO, QSB, Mathematica, etc. have inbuilt functions to solve the non-linear programming problems. Solving problem using these

software, we can save a lot of computation time. The Problem (P22) is solved in the LINGO software [52] and the solution gives the optimal release time $T^* = 84.45$ h. Degree of aspiration of the management goals is $\alpha = 0.6935$. The total amount of testing resources spent by the optimal release time is $C(T^*) = \$1880.66$ and the achieved level of failure intensity is $\lambda(T^*) = 0.001292$. The risk cost of failure in field is \$34.43, which implies the achieved level of reliability $R(T^*) = 0.9885$. The optimal solution of Problem (P22) solves Problem (P18). Here it can be noted that the fuzzy optimization method provides sub-optimal solution due to the subjective nature of the method. However since the method provides huge amount of flexibility to the management in decision making, it is widely used.

The constraints corresponding to cost and intensity function in this problem are the two important objectives in the SRTD problem, which may have different relative importance. We can assign weights to the cost and intensity membership constraints and use the weighted min–max approach to solve the problem (problem P**). It is reasonable to assign a higher weight to the cost objective. If $w = \{0.6, 0.4\}$ be the weights assigned to the two objectives, we obtain the optimal release time $T^* = 85.18$ h and $\alpha = 0.4125$. The total amount of testing resources spent is $C(T^*) = \$1881.26$ and the achieved level of failure intensity is $\lambda(T^*) = 0.00119$. The risk cost of failure in field is \$31.74, which implies the achieved level of reliability $R(T^*) = 0.9894$. We can see that this solution is more acceptable in terms of risk cost, achieved level of failure intensity and reliability. The risk cost decreases by amount \$2.69 and failure intensity and reliability levels improve by amounts 0.0001 and 0.0009, respectively. However the cost and release time have increased by amount \$0.585 and 0.73 h, respectively. Similarly we can solve the problem for different values of weights. Table 10.5 summarizes some alternative solution of the problem for different values of the weights.

From the table we can see that if a higher weight is assigned to the cost function, risk cost decreases and there is an improvement in the achieved level of failure intensity, reliability and α, however the total cost and release time increase. On the other hand the situation is vice versa if a lower weight is assigned to the cost function. This flexible solution methodology provides us the various alternative solutions for the problem to choose from as well as a method for achieving maximum possible level of management goals. This is another reason for studying the optimization problems under fuzzy environment.

Table 10.5 Solution of Problem (P15) for different values of weights

Weights	Release time (in hrs)	α	Cost (in \$)	Failure intensity	Risk cost (in \$)	Reliability
(0.5,0.5)	84.45	0.6935	1880.66	0.00129	34.43	0.9885
(0.6,04)	85.18	0.4125	1881.26	0.00119	31.74	0.9894
(0.7,0.3)	86.47	0.47	1882.78	0.00103	27.48	0.9908
(0.4,0.6)	83.98	0.2784	1881.41	0.00136	36.25	0.9879

Exercises

1. Reliability, scheduled delivery and cost are the three main quality attributes for almost all software. How a release time optimization problem handles these attributes of software while determining the optimal release time?
2. Before any software development process is realized, the management mostly decide the schedule for software delivery. Even then software reliability engineering principles say optimally determine the release time. Comment.
3. Explain the importance of soft computing principles and techniques for formulating and solving release time problems.
4. The simplest cost model in release time determination is

$$C(T) = C_1 m(T) + C_2(m(T_{lc}) - m(T)) + C_3 T.$$

   The cost model has three components. Give interpretations for each of the components, how they handle the various concerns related to the release time determination.
5. In Sect. 10.2.2 a release policy is formulated using a cost model that includes the penalty cost due to late delivery. The solution is discussed for an s-shaped SRGM. Using the same cost function and the exponential SRGM $m(t) = a(1 - e^{-bt})$ derive the solution of the release time problem.
6. Release policy discussed in Sect. 10.2.8 for a pure error generation fault complexity based SRGM is formulated to minimize the total expected cost. Reformulate and solve the problem adding the constraints to the number of remaining faults of each type in the system before its release.
7. Using the data given in Application 10.2 determine the release time for the policy formulated in exercise 5.
8. Given the fuzzy release time problem, which minimizes the fuzzy risk cost (RC) subject to fuzzy total cost, fuzzy failure intensity constraint and non-negativity restriction constraint formulated on the Erlang SRGM describing three levels of fault complexity.

$$\begin{aligned}
&\text{Minimize} \quad RC = \tilde{C}_4(1 - R(x/T)) \\
&\text{Subject to } \tilde{C}(T) = \left\{ \begin{aligned} &\tilde{C}_{11}m_1(T) + \tilde{C}_{21}m_2(T) + \tilde{C}_{31}m_3(T) \\ &\qquad + \tilde{C}_{12}(m_1(T_{lc}) - m_1(T)) \\ &\qquad + \tilde{C}_{22}(m_2(T_{lc}) - m_2(T)) \\ &\qquad + \tilde{C}_{32}(m_3(T_{lc}) - m_3(T)) + \tilde{C}_3 T \end{aligned} \right\} \lesssim Z \\
&\qquad \lambda(T) \lesssim \tilde{\lambda}_0 \\
&\qquad T \geq 0
\end{aligned}$$

where all notations have their usual meaning, Z is the total budget and $\tilde{\lambda}_0$ is the desired failure intensity.

$$m(t) = m_1(t) + m_2(t) + m_3(t)$$
$$= \begin{pmatrix} ap_1(1 - e^{-b_1 t}) + ap_2(1 - (1 + b_2 t)e^{-b_2 t}) \\ +ap_3\left(1 - \left(1 + b_3 t + \left(b_3^2 t^2/2\right)\right)e^{-b_3 t}\right) \end{pmatrix};$$
$$ap_1 + ap_2 + ap_3 = a$$

Use the fuzzy optimization technique to determine the release time of the software. The following data are given:

The software was tested for 12 weeks during which 136 failures were reported. The faults were categorized as 55 simple, 55 hard and 26 complex with respect to time in isolating and removing them after their detection. The estimated values of parameters are $a = 180$, $b_1 = 0.12667$, $b_2 = 0.21499$ and $b_3 = 0.41539$. Use the defuzzification function $F_2(A) = (a_l + 2a + a_u)/4$. The fuzzy cost coefficients along with permissible tolerance level of failure intensity and budget are specified as Triangular Fuzzy Numbers in the following table.

Fuzzy parameter (A)	a_1 (in \$)	a (in \$)	a_u (in \$)
C_{11}	14.4	15.1	15.4
C_{21}	17	18	19
C_{31}	20.5	21.5	24.5
C_{12}	23	24.5	28
C_{22}	31	36	37
C_{32}	46	48	58
C_3	74	81.5	83
C_4	17000	20750	21500
λ	0.0008	0.00105	0.0011
Z	8600	8650	8900

References

1. Taha HA (2006) Operations research: an introduction, 8th edn. Prentice Hall, India
2. Okumoto K, Goel AL (1980) Optimum release time for software systems based on reliability and cost criteria. J Syst Softw 1:315–318
3. Yamada S, Osaki S (1987) Optimal software release policies with simultaneous cost and reliability requirements. Eur J Oper Res 31:46–51
4. Kapur PK, Garg RB (1989) Cost-reliability optimum release policies for software system under penalty cost. Int J Syst Sci 20:2547–2562
5. Kapur PK, Garg RB (1990) Optimal software release policies for software reliability growth models under imperfect debugging. Recherché Operationanelle/Oper Res 24:295–305

6. Kapur PK, Garg RB (1991) Optimal release policies for software systems with testing effort. Int J Syst Sci 22(9):1563–1571
7. Kapur PK, Garg RB (1992) A software reliability growth model for an error removal phenomenon. Softw Eng J 7:291–294
8. Yun WY, Bai DS (1990) Optimum software release policy with random life cycle. IEEE Trans Reliab 39(2):167–170
9. Kapur PK, Bhalla VK (1992) Optimal release policy for a flexible software reliability growth model. Reliab Eng Syst Saf 35:49–54
10. Kapur PK, Garg RB, Bhalla VK (1993) Release policies with random software life cycle and penalty cost. Microelectron Reliab 33(1):7–12
11. Kapur PK, Agarwal S, Garg RB (1994) Bi-criterion release policy for exponential software reliability growth models. Recherche Operationanelle/Oper Res 28:165–180
12. Kapur PK, Xie M, Garg RB, Jha AK (1994) A discrete software reliability growth model with testing effort. In: Proceedings 1st International conference on software testing, reliability and quality assurance (STRQA), 21–22 December 1994, New Delhi, pp 16–20
13. Pham H (1996) A software cost model with imperfect debugging, random life cycle and penalty cost. Int J Syst Sci 27:455–463
14. Pham H, Zhang X (1999) A software cost model with warranty and risk costs. IEEE Trans Comp 48(1):71–75
15. Huang CY, Kuo SY, Lyu MR (1999) Optimal software release policy based on cost and reliability with testing efficiency. In: Proceedings 23rd IEEE annual international computer software and applications conference, Phenoix, AZ, pp 468–473
16. Huang CY, Lo JH, Kuo SY, Lyu MR (1999) Software reliability modeling and cost estimation incorporating testing-effort and efficiency. In: Proceedings 10th international symposium software reliability engineering (ISSRE'1999), pp 62–72
17. Huang CY (2005) Cost reliability optimal release policy for software reliability models incorporating improvements in testing efficiency. J Syst Softw 77:139–155
18. Huang CY, Lyu MR (2005) Optimal release time for software systems considering cost, testing effort and test efficiency. IEEE Trans Reliab 54(4):583–591
19. Kapur PK, Gupta A, Jha PC (2007) Reliability growth modeling and optimal release policy of a n-version programming system incorporating the effect of fault removal efficiency. Int J Autom Comput 4(4):369–379
20. Kapur PK, Gupta A, Gupta D, Jha PC (2008) Optimum software release policy under fuzzy environment for a n-version programming system using a discrete software reliability growth model incorporating the effect of fault removal efficiency. In: Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex systems, Macmillan Advance Research Series, pp 803–816
21. Jha PC, Gupta D, Gupta A, Kapur PK (2008) Release time decision policy of software employed for the safety of critical system under uncertainty. OPSEARCH. J Oper Res Soc India 45(3):209–224
22. Gupta A (2009) Some contributions to modeling and optimization in software reliability and marketing. Ph.D. thesis, Department of OR, Delhi University, Delhi
23. Rommelfanger HJ (2004) The advantages of fuzzy optimization models in practical use. Fuzzy Optim Decis Mak 3:295–309
24. Gupta CP (1996) Capital budgeting decisions under fuzzy environment. Financ India 10(2):385–388
25. Xiong Y, Rao SS (2004) Fuzzy nonlinear programming for mixed-discrete design optimization through hybrid genetic algorithm. Fuzzy Sets Syst 146:167–186
26. Zadeh LA (1965) Fuzzy sets. Inf Control 8:338–353
27. Zimmermann H J (1991) Fuzzy set theory and its applications. Academic Publisher, New York
28 Lee KH (2005) First course on fuzzy theory and applications. Springer, Berlin. doi: 10.1007/3-540-32366-X

29. Ramik J (2001) Soft computing: overview and recent developments in fuzzy optimization. Research Report, JAIST Hokuriku
30. Bellman RE, Zadeh LA (1973) Decision making in a fuzzy environment. Manage Sci 17:141–164
31. Tiwari RN, Dharmar S, Rao JR (1987) Fuzzy goal programming—an additive model. Fuzzy Sets Syst 24:27–34
32. Mohamed RH (1997) The relationship between goal programming and fuzzy programming. Fuzzy Sets Syst 89:215–222
33. Sandgren E (1990) Nonlinear integer and discrete programming in mechanical design optimization. ASME J Mech Des 112:223–229
34. Tang J, Wang D (1996) Modeling and optimization for a type of fuzzy nonlinear programming problems in manufacturing systems. In: Proceeding 35th IEEE conference on decision and control, pp 4401–4405
35. Guan XH, Liu WHE, Papalexopoulos AD (1995) Application of a fuzzy set method in an optimal power flow. Elect Power Syst Res? 34:11–18
36. Xiang H, Verma BP, Hoogenboom G (1994) Fuzzy irrigation decisions support system. In: Proceedings 12th national conference on artificial intelligence, Part 2(2), Seattle, WA
37. Kuntze HB, Sajidman M, Jacubasch A (1995) Fuzzy-logic concept for highly fast and accurate position control of industrial robots. In: Proceedings 1995 IEEE international conference on robotics and automation, Part 1(3), pp 1184–1190
38. Sousa JM, Babuska R, Verbruggen HB (1997) Fuzzy predictive control applied to an air-conditioning system. Control Eng Prac 5(10):1395–1406
39. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R 28(3):206–211
40. Wood A (1996) Predicting software reliability. IEEE Comp 11:69–77
41. Yamada S, Osaki S (1985) Discrete software reliability growth models. Appl Stoch Models Data Anal 1:65–77
42. Yamada S, Ohba M, Osaki S (1983) S-shaped software reliability growth modeling for software error detection. IEEE Trans Reliab R 32(5):475–484
43. Yamada S, Ohtera H, Narihisa H (1986) Software reliability growth models with testing-effort. IEEE Trans Reliab R 35:19–23
44. Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York. ISBN 0-07-044093-X
45. Xie M, Yang B (2003) A study of the effect of imperfect debugging on software development cost. IEEE Trans Softw Eng 29(5):471–473. doi:10.1109/TSE.2003.1199075
46. Ohba M, Chou XM (1989) Does imperfect debugging effect software reliability growth. In: Proceedings 11th international conference of software engineering, pp 237–244
47. Kapur PK, Gupta D, Gupta A, Jha PC (2008) Effect of introduction of fault and imperfect debugging on release time. J Ratio Math 18:62–90
48. Pham H, Zhang X (2003) NHPP software reliability and cost models with testing coverage. Eur J Oper Res 145(2):443–454
49. Kapur P K, Garg RB, Aggarwal A G, Tandon A (2010) General framework for change point problem in software reliability and related release time problem. In: Proceedings ICQRIT 2009
50. Kapur PK, Bai M, Bhushan S (1992) Some stochastic models in software reliability based on NHPP. In: Venugopal N (ed) Contributions to stochastics, Wiley, New Delhi
51. Bector CR, Chandra S (2005) Fuzzy mathematical programming and fuzzy matrix games. Springer, Berlin
52. Thirez H (2000) OR software LINGO. Eur J Oper Res 124:655–656

Chapter 11
Allocation Problems at Unit Level Testing

Notation

$pr\ \{.\}$	Probability
$\{N(t),\ t \geq 0\}$	Counting process representing the cumulative number of software faults detected in the time interval [0, t]
$m(t)$	Mean value function in the NHPP model, $m(0) = 0$
$m_f(t)$	Mean value function of the failure process in the NHPP model, $m_f(0) = 0$
$m_r(t)$	Mean value function of the removal process in the NHPP model, $m_r(0) = 0$
$w(t)$, $W(t)$	Current testing-resource expenditures at testing time t for ($w_i(t)$, $W_i(t)$) software (module i) and its integral form, i.e., $W(t) = \int_0^t w(x)dx; \quad W_i(t) = \int_0^t w_i(x)dx,$
$a(a_i)$	Expected initial fault content in software (module i), a > 0
b (b_i)	Constant fault detection/removal rate per remaining faults in software (module i) $0 < b < 1$
i	Subscript for each software-module number $i = 1, 2, \ldots, N$
v_i	Weighted importance for each software module, $v_i > 0$
z_i, Z	The number of software faults remaining in each software module and the whole system
W	The specified total testing-resource expenditures before module testing, $W > 0$
$R(s)$	Software reliability which means that no software failures occur during the time interval $(0, s](s \geq 0)$ after the testing process
γ	Constant parameter related to the failure rate, $\gamma > 0$
R_0	Objective value of the software reliability, $0 < R_0 < 1$
*	Superscript that denotes the optimal solution of the test resource allocation problem
A_i	$v_i a_i b_i$ (detectability of module i)
L, λ	Lagrangian, Lagrange multiplier λ

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_11,
© Springer-Verlag London Limited 2011

11.1 Introduction

As mentioned in previous chapter reliability, scheduled delivery and cost are the three main quality attributes for almost all software. By determining the releases time of the software optimally taking into consideration the various constraints and aspects of the software enables to best achieve these objectives. Many a time in the software release time problems we have seen keeping the cost minimization objective alone may leave us with a solution that the reliability achieved is low. On the other hand reliability maximization objective alone may require large budget. The two objectives simultaneously are conflicting and demands bounds on budget and achievable reliability. The release time problem by no means controls the consumption of the testing resources. The total software testing cost as well as reliability depends largely on the consumption of the testing resources during the testing process. Judicious consumption of the testing resources can enable us to achieve much higher reliability in the same expenditure or even less. It has direct impact on the software release decision, the quality level of the software achieved and the cost incurred. Hence before one builds a model for software release time one should determine how the testing resources should be allocated in the different levels of testing and software components. The time-dependent use of testing resources can be observed as a consumption curve of testing resources.

In the software reliability literature this problem is widely studied as the problem of allocating the limited testing resources among the different modules of the software so as to optimize some performance index (such as maximize reliability or number of faults removed, minimize failure intensity or cost, etc.) with respect to the software. Such optimization problems are known as *Resource Allocation Problems*. More specifically the problem can be explained as follows.

Software life cycle consists mainly of the following phases: requirement and specification, design, coding, testing and operations/maintenance. During the testing stage software is tested and detected faults are corrected. Most of the real life software are designed based on modular structure. Each module or a group of modules together are capable of performing some function of the software. The testing process for such software usually consists of three stages—module testing, integration testing and acceptance testing. In module testing each module is tested independently for its intended function. In integration testing, all the software modules are interconnected according to predetermined logical structure and the whole software system is tested. In acceptance testing the software system is tested by customers or is tested using the test sets supplied by the customer. Testing resources get consumed in each of them. The problem of testing-resource allocation has mainly two concerns firstly how much testing resources are to be allocated to each of the testing stages. Secondly how much testing resources should be allocated to each of the modules so that the software performance can be optimized measured in terms of reliability, number of remaining faults, failure intensity, total resources consumed, etc. The studies in the literature are mainly concerned with the second problem. Allocation of

testing resources among the testing stages is mainly based on expert opinion, past project data, or sometimes independent budget for the different stages is kept. In case of resource allocation at unit testing level one who is not aware of why such an allocation is required may say that allocate equal resources to each module, as every module, is an integrated part of the software. But it maynot be an optimal policy. Although, each module has its unique importance in the software but all of them may not be equally important. Some can provide major functionality to the software while some may only be supporting some functions. Some modules might be frequently called or used while some can have rare or middle order calling frequency. Some can be very large in size while others may not be so. During testing the detectability of faults in some modules can be high while low in others. Some modules can have only simple types of faults while faults in others can have varying degrees of complexity. The complexity level of faults with a module may also vary. Like that we can face a number of different situations in unit testing. It requires an appropriate measurement of the testing process progress as well as judicious allocation of the testing resources among the modules to achieve an overall high level of reliability. Therefore the software project manager should monitor the testing process closely and effectively allocate the resources in order to reduce the testing cost and to meet the given reliability requirements.

All the testing activities of different modules should be completed within a limited time, and these activities normally consume approximately 40–50% of the total amount of limited software development resources [1]. Typically, module testing is the most time-critical part of testing to be performed. Therefore, project managers should know how to allocate the specified testing-resources among all the modules. Scope of this chapter is restricted to resource allocation problem for module testing (unit testing) level. It may be noted that most of the allocation problems studied in the literature are related to the allocation of testing resources to each of the modules at the module testing level considering each module to be independent of each other. However dependency of modules can also be considered with ease if we optimize the resource allocation at the system testing level when on an input a sequence of modules are called to get the desired output. At the module testing level modules can be considered to be independent of each other since they are designed independently. Since the allocation problem discussed in this chapter considers the testing at the modular level, we therefore consider the modules to be independent of each other. Each module may contain different number of faults and that of different severity. Hence fault detection/removal phenomenon in modules can be represented through distinct SRGM. Throughout this book we have discussed a number of SRGM and their applications on real life data sets. In this chapter we will discuss how these models can be used to depict the reliability growth of independent modules during unit testing and using this information we will build optimization models to determine the optimal allocation of testing resources to the software modules so that the software performance can be optimized.

11.2 Allocation of Resources based on Exponential SRGM

Ohtera and Yamada [1] were the first to discuss two management problems to achieve a reliable software system efficiently during module testing stage in the software development process by applying NHPP-based software reliability growth model [2]. The relationship between the testing resources spent during the module testing and the detected software faults can be described by the test effort based SRGM. The software development manager has to decide how to use the specified testing resources effectively in order to maximize the software quality measured in terms of reliability. That is, to develop the reliable software system, the manager must allocate the specified amount of testing- resource expenditures for each software module. Two kinds of testing-resource allocation problems are considered to make the best use of a specified total testing-resource expenditure in module testing. The manager has to allocate it appropriately to each software module which is tested independently and simultaneously.

11.2.1 Minimizing Remaining Faults

Based on the test effort based exponential software reliability growth model (refer Sect. 2.7) the testing-resource allocation problem is formulated under the following assumptions

1. The software system is composed of N independent modules. The number of software faults remaining in each module can be estimated from the test effort based exponential software reliability growth model.
2. Each software module is subject to failure at random times caused by the faults remaining in it.
3. If any of the software modules fails, the software system fails.
4. The failure process of the software module i is modeled by a non-homogeneous Poisson process with mean value function $m_i(t)$.
5. The total amount of testing-resource expenditures for the module testing is specified.
6. The manager has to allocate the specified total testing-resource expenditures to each software module so that the number of software faults remaining in the system may be minimized.

Following the SRGM in Sect. 2.7 the mean value function of the SRGM [3] for module i is given as

$$m_i(t) = a_i\left(1 - \mathrm{e}^{-b_i W_i(t)}\right); \quad i = 1, 2, \ldots, N \tag{11.2.1}$$

and the expected number of software faults remaining in ith module is thus given as

$$z_i(t) = a_i - m_i(t) = a_i \mathrm{e}^{-b_i W_i(t)}; \quad i = 1, 2, \ldots, N \tag{11.2.2}$$

Any software cannot be tested indefinitely to detect/remove all the faults lying in the software, since the software has to be released in the market or to the specified user for a project kind of software at a predefined software release time. Hence software-testing time is almost fixed (say T). Therefore, without any loss of generality the number of faults removed by time T can be assumed to be a function of testing effort explicitly in Eq. (11.2.2). So if W_i be the testing effort that has to be spent on the ith module during testing time T, the expected number of software faults remaining in ith module can be rewritten as

$$z_i(W_i) = a_i - m_i(t) = a_i e^{-b_i W_i}; \quad i = 1, 2, \ldots, N \tag{11.2.3}$$

If v_i is the weighting factor to measure the relative importance of ith module, the testing resource allocation problem that minimizes the expected total faults remaining in all modules is formulated as

$$\begin{aligned} &\text{Minimize} \quad Z = \sum_{i=1}^{N} v_i a_i . e^{-b_i W_i} \\ &\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0 \quad i = 1, 2, \ldots N. \end{aligned} \tag{P1}$$

For solving such a problem first one must determine the unknown parameters of the SRGM, a_i and b_i either using real software failure data of some previous period or similar software. Assuming that these unknowns have already been estimated using some real life data (refer Sect. 2.9) the above mentioned problem is solved by the method of Lagrange multiplier. Consider the following Lagrangian for the problem (P1)

$$L = \sum_{i=1}^{N} v_i a_i . e^{-b_i W_i} + \lambda \left(\sum_{i=1}^{N} W_i - W \right), \tag{11.2.4}$$

and the necessary and sufficient conditions [11] for the minimum are

$$\begin{aligned} &\frac{\partial L}{\partial W_i} = -v_i a_i b_i e^{-b_i W_i} + \lambda \ge 0, \\ &W_i . \frac{\partial L}{\partial W_i} = 0; \quad i = 1, 2, \ldots, N, \\ &\sum_{i=1}^{N} W_i = W; \quad W_i \ge 0 \quad i = 1, 2, \ldots, N. \end{aligned} \tag{11.2.5}$$

Without loss of generality, we can assume that the following condition is satisfied for the modules

$$A_1 \ge A_2 \ge \ldots \ge A_{K-1} \ge A_{K+1} \ge \ldots \ge A_N \tag{11.2.6}$$

This means modules are arranged in order of fault detectability. Now, if $A_k \geq \lambda \geq A_{k+1}$, from (11.2.5) we have

$$W_i = \max\left\{0, \frac{1}{b_i}(\ln A_i - \ln \lambda)\right\};$$

i.e.,

$$\begin{aligned} W_i &= \frac{1}{b_i}(\ln A_i - \ln \lambda) \quad i = 1, 2, \ldots, k, \\ W_i &= 0 \quad i = k+1, \ldots, N. \end{aligned} \tag{11.2.7}$$

From (11.2.5) and (11.2.7), $\ln \lambda$ is given by

$$\ln \lambda = \left(\sum_{i=1}^{k} \frac{1}{b_i} \ln A_i - W\right) \Bigg/ \left(\sum_{i=1}^{k} \frac{1}{b_i}\right) \quad k = 1, 2, \ldots, N \tag{11.2.8}$$

Let λ_k denote the value of the right-hand side of (11.2.8). Then, the optimal Lagrange multiplier λ^* exists in the set $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$. Hence, we can obtain λ^* by the following algorithm.

Algorithm 11.1

(i) Set $k = 1$.
(ii) Compute λ_K by (11.2.8).
(iii) If $A_k > \lambda_k \geq A_{k+1}$, then $\lambda^* = \lambda_k$ (stop). Otherwise, set $k = k + 1$ and go back to (ii).

The optimal solution $W_i^*(i = 1, 2, \ldots, N)$ is given by

$$W_i^* = \frac{1}{b_i}(\ln A_i - \ln \lambda^*) \quad i = 1, \ldots, k \tag{11.2.9}$$

$$W_i^* = 0 \quad i = k+1, \ldots, N \tag{11.2.10}$$

11.2.2 Minimizing Testing Resource Expenditures

The previous problem minimizes the faults remaining in the software in each module in order to attain maximum reliability subject to the resource availability constraint. This problem is formulated to minimize the total testing-resource expenditure in module testing such that the number of software errors remaining in the system is Z at the termination of module testing again assuming N independent modular structure.

The problem is formulated as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} W_i = W \\ \text{Subject to} \quad & Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i W_i} \\ & W_i \geq 0 \quad i = 1, 2, \ldots, N \end{aligned} \tag{P2}$$

Again preceding in the same way as in case of problem (P1) the Lagrange is formulated

$$L = \sum_{i=1}^{M} W_i + \lambda \left(\sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i W_i} - Z \right) \tag{11.2.11}$$

Solving the above Lagrange the optimal solution W_i^* is obtained

$$W_i^* = \left[\ln \left(\frac{A_i}{Z} \sum_{j=1}^{k-1} \frac{1}{b_j} \right) \right] \Big/ b_i, \quad i = 1, 2, \ldots, k-1 \tag{11.2.12}$$

$$W_i^* = 0, \quad \text{otherwise} \tag{11.2.13}$$

11.2.3 Dynamic Allocation of Resources for Modular Software

The allocation problems discussed by Ohtera and Yamada [1] and Yamada et al. [4] considered (1) minimization of the mean number of remaining faults in the modules when a fixed amount of resources is given and (2) minimization of the required amount of resources while the mean number of remaining faults in the modules is equal to a given requirement. In these problems, only the mean number of remaining faults in the modules was considered. However, even when a fixed amount of testing resources is allocated to a software module, the number of faults that can be detected in this module is not fixed. Consider an example that after a certain period of testing of modular software the decision maker estimates the remaining fault content in the software and the fault detection rate. Using this information the allocation of the testing resources is determined based on problem (P1) under a specified budget. If we assume the total amount of allocated resources is spend uniformly over the remaining testing period the number of detected faults may vary from the expected values, due to the random nature of the fault detection process. Also the fault detection process is not solely determined by the testing resources consumption, rather there are a number of factors that influence the testing process such as test case coverage, defect density, fault dependencies etc. Hence when software-module testing is completed, the actual number of

remaining faults in the modules may turn out to be much larger than the expected one. To reduce this possibility, we should reduce the variance of the number of remaining faults in the software modules.

Leung [5] proposed a dynamic resource allocation strategy for software-module testing, which provides a method to reduce this variance. This strategy considers the number of faults detected in each module as module testing is proceeding, re-estimates the model parameters using all the available fault detection data and adjusts the resource allocation dynamically.

The policy can be explained in detail as follows
Divide the total testing time for software module testing into K testing periods, of say 1-week duration each. Duration of time period may or may not be fixed. The software project manager first records the fault detection times and the total number of detected faults in each software module in testing periods. At the end of this testing period, the project manager selects a model to represent the testing process, uses the data recorded in the first period to estimate the unknown parameters of this model and then determines the mean number of remaining faults for each software module. With these estimates, he now determines the amount of resources that should be allocated to each software module in the next testing period. The above process is repeated for the K time periods.

Allocation of testing resources following the above policy can reduce the final fault content variance. Consider the following example, at the end of a testing period, suppose the mean number of remaining faults in module 1 is large while the mean number of remaining faults in module 2 is small. The project manager will allocate more testing resources to module 1 but fewer resources to module 2 in the next testing period. After several testing periods, if the mean number of remaining faults in module 1 becomes relatively small but that in module 2 becomes relatively large, then the project manager allocates less testing resources to module 1 and allocates more testing resources to module 2. By taking into account the variations of the number of detected faults during testing and re-allocating resources to the software modules in each testing period, the variance of the number of remaining faults in software module 1 or software module 2 at the end of software-module testing can be reduced.

Leung [5] explained this allocation procedure with respect to the allocation policies discussed by Ohtera and Yamada [1]. In the jth testing period which starts at time T_j and ends at T_{j+1}, the mean number of remaining faults at the beginning of the jth testing period is

$$z_{ij} = a_{ij}\mathrm{e}^{-b_{ij}W_{ij}} \tag{11.2.14}$$

where a_{ij} is the remaining fault content, b_{ij} is the fault detection rate and W_{ij} is the total amount of resources allocated to module i in the jth testing period. If we assume resources are spent uniformly over the testing period then

$$W_{ij} = w_{ij}(T_{j+1} - T_j) \tag{11.2.15}$$

Now to determine resource allocation $W_{ij} = w_{ij}(T_{j+1} - T_j)$ reconsider the problem (P1).

11.2.4 Minimize the Mean Fault Content

Given the total amount of available resources W, let the amount of resources W_j be expendable in the jth testing period to N modules. Allocation of these resources among the N modules can be done according to the following model

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} v_i a_{ij} e^{-b_{ij} W_{ij}} \\ \text{Subject to} \quad & W_{ij} \geq 0 \\ & \sum_{i=1}^{N} W_{ij} = W_j \end{aligned} \tag{P3}$$

If we ignore the subscript j the problem (P3) is same as (P1) and can be solved using Algorithm 11.1. Now follow the sequential steps given in Algorithm 11.2 to dynamically allocate the testing resources to N modules in K testing periods after observing the testing period for certain time initially.

Algorithm 11.2

1. Set $j \leftarrow 1$.
2. Estimate the parameters b_{ij} and a_{ij} for module i for $i = 1, 2, \ldots, N$ from the recorded data. Note: In case the decision is to be made before the start of the testing process let $W_{i0} \geq 0$ units of testing resources be allocated to software module i, and using the estimates of the model parameters determined from some earlier similar project or expert judgment generate a fault detection process for each software module. Using this generated testing data now estimate b_{i1} and a_{i1}, $i = 1, \ldots N$.
3. Calculate the optimal resource allocation $\{W_{ij}^*, i = 1, 2, \ldots, N\}$ for the jth testing period. We can assume that $W_j = \left(\frac{T_{j+1} - T_j}{T_{k+1} - T_1}\right) W$.
4. Start the jth testing period.
5. Record the total number of detected faults X_{ij} and the fault detection times $t_1^{(ij)}, t_2^{(ij)}, \ldots, t_{X_{ij}}^{(ij)}$ for software module i, $i = 1, 2, \ldots, N$.
6. At the end of the jth testing period, if $j < K$, then $j \leftarrow j + 1$ and go to step (2), otherwise stop.

The model parameters b_{ij} and a_{ij} are re-estimated at the end of each testing period based on all the available fault detection data. Therefore, in each step better estimation accuracy is achieved as more data is available and estimates are based on all available data. It makes our resource allocation more efficient and judicious.

To measure the relative accuracy improvement, we define the relative estimation error for any software module (say module 1) to be

$$\text{Relative estimation error} = \frac{|E_1|}{|E|}$$

where E is the estimation error when all the available fault detection data are used in estimation and E_1 is the estimation error when only the fault detection data recorded before software module testing are used. If the relative estimation error is smaller, then the estimation accuracy improvement is larger. Similar dynamic allocation policy is formed for the problem (P2) which minimizes the total testing-resource consumption (for details refer Leung [5]). In fact we can use the similar steps to dynamically allocate the testing resources for any testing process represented by any existing SRGM.

Application 11.1

The model finds its application in determining the amount of the testing resources to be allocated to the different modules of the software during module/unit testing stage. Consider software with ten independent modules. Given that each software module has already been tested for some time and test effort based exponential SRGM is applied on each module to estimate failure process parameters. The estimates of the parameters of the SRGM [a_i, b_i (in 10^{-4})] for each module are tabulated in Table 11.1. The weights assigned to each module are also tabulated in the table. The problem is solved with total resources W = 50,000 units. The allocations made to each module as well as the remaining software faults in each module are also tabulated in the table.

From the table we can calculate that the total software faults remaining are $Z = 162$, whereas before the start of the testing phase the fault content was 442. It implies that now 63.35% fault content can be reduced from the software when a total of 50,000 units of testing resources are consumed. Since the whole of the available testing resources gets consumed in this allocation, if we further want to minimize the remaining fault content in the unit testing we should increase the amount of testing resources. For this purpose we can apply the testing effort control problem as discussed in Chap. 5.

Table 11.1 Data and allocated testing resources W_i^* for problem (P1)

Module	a_i	v_i	b_i ($\times 10^{-4}$)	W_i^*	z_i
1	89	1	4.1823	6516.2	5.8
2	25	1	5.0923	3244.9	4.8
3	27	1	3.9611	3731.6	6.2
4	45	1	2.2956	6287.7	10.6
5	39	1	2.5336	5521.6	9.6
6	39	1	1.7246	5881.5	14.1
7	59	1	0.8819	8590.8	27.7
8	68	1	0.7274	9719.4	33.5
9	37	1	0.6824	506.2	35.7
10	14	1	1.5309	0.0	14.0

Table 11.2 Testing resource allocation based on problem (P2)

Module	W_i^*	z_i
1	7699.6	3.6
2	4216.8	2.9
3	4981.1	4.0
4	8443.9	6.5
5	7475.2	5.9
6	8751.5	8.6
7	14203.2	16.9
8	16523.9	20.5
9	7759.4	21.8
10	2388.4	9.7

As an alternative we can apply the problem (P2) on the data to determine how much minimum testing efforts are required and how to allocate these resources among the modules to achieve a particular level of reliability. Remaining fault content is a determinant of the software reliability. We may specify that we want to terminate the unit testing when the total remaining fault content equals 100 in the software instead of 162 as given by the solution of the problem (P1) for testing resources 50,000. In this case we apply problem (P2) to the data [a_i, b_i (in 10^{-4}), v_i] given in Table 11.1. The optimal allocation of testing resources and the remaining fault content according to problem (P2) is tabulated in Table 11.2.

Then, the total amount of testing resource consumed in the module testing is $W = \sum_{i=1}^{10} W_i^* = 82,443$ units. Comparison of the results of problems (P1) and (P2) suggests that an extra amount of testing-resource expenditures equals 32,443 (= 82,443 − 50,000) units needs to be spent on testing to reach the remaining fault content of 100.

11.2.5 Minimizing Remaining Faults with a Reliability Objective

Section 11.2.1 describes a resource allocation problem with remaining fault minimization objective, under the limited testing resources. Sometimes the decision maker may not be satisfied with the results obtained with this model. One major reason for this dissatisfaction could be that the reliability level achieved following the allocation made according to problem (P1) is not matching the decision maker's aspiration from the testing process. On the other hand due to the resource allocation made according to formulation (P1), (P2) and (P3) some of the modules may remain untested due to very hard detectability of faults (i.e. modules having very low values of A_i's may not get any allocation of resources). In order to consider the level of reliability that we may achieve from testing using the specified amount of testing resources, we can modify the problem (P1) to include a reliability aspiration constraint for each module and guarantee a

certain level of reliability. It also ensures that each module will be tested so that a minimum level of reliability is achieved for each. The problem was formulated by Yamada et al. [4]. Keeping the Assumptions 1–5 of problem (P1) and modifying the sixth as follows the problem is reformulated.

Assumption 6 modified

6. We need to allocate the amount of testing-resource expenditures to each module so that the attained software reliability after the testing is greater than or equal to a reliability objective, say R_0

Defining reliability

$$R(s) = \mathrm{e}^{-\gamma z(t)s} \tag{11.2.16}$$

the modified problem is

$$\begin{aligned} &\text{Minimize} \quad Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i W_i} \\ &\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0 \quad i = 1, 2, \ldots N \end{aligned} \tag{P4}$$

$$R(s) = \mathrm{e}^{-\gamma_i a_i . s \mathrm{e}^{-b_i W_i}} \ge R_0 \quad i = 1, 2, \ldots, N \tag{11.2.17}$$

Equations (11.2.17) put a reliability aspiration constraint on each module with aspiration level R_0. From this we can see that

$$W_i \ge -\frac{1}{b_i} \ln\left(-\frac{\ln R_0}{\gamma_i a_i s}\right) \tag{11.2.18}$$

The right-hand side of (11.2.18) is constant for each module, let us denote it by d_i. The reliability aspiration constraints can thus be transformed as

$$W_i \ge c_i, \quad \text{where} \quad c_i = \max\{0, d_i\}; \quad i = 1, 2, \ldots N \tag{11.2.19}$$

i.e. we obtain the following transformed optimal testing-resource allocation problem

$$\begin{aligned} &\text{Minimize} \quad Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i W_i} \\ &\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0 \quad i = 1, 2, \ldots N \\ &\qquad\qquad W_i \ge c_i \quad i = 1, 2, \ldots N \end{aligned} \tag{P4.1}$$

Denote $x_i = W_i - c_i$, so the problem (P4.1) is further transformed as

$$\begin{aligned} &\text{Minimize} \quad Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i c_i} \mathrm{e}^{-b_i x_i} \\ &\text{Subject to} \quad \sum_{i=1}^{N} x_i \le W - \sum_{i=1}^{N} c_i, \quad x_i \ge 0, \quad i = 1, 2, \ldots, N \end{aligned} \tag{P4.2}$$

Hence the problem under consideration reduces to the optimal testing-resource allocation problem (P1) if we make the following transformations

$$W \leftarrow W - \sum_{i=1}^{N} c_i; \quad v_i \leftarrow v_i e^{-b_i c_i}; \text{ and } W_i \leftarrow x_i \tag{11.2.20}$$

Huang et al. [6] reformulated the above problem defining the reliability at time t as the ratio of the cumulative number of detected faults at time t to the expected number of initial faults, i.e.

$$R(t) \equiv \frac{m(t)}{a}$$

and followed the similar steps to compute the optimal resource allocation. Along with this the authors have also discussed the allocation problem of minimizing the amount of testing effort given the number of remaining faults and a reliability objective. We will discuss this problem after discussing an application of problem (P4).

Application 11.2

For the application of the problem (P4) on a real life project consider software with ten modules. The information tabulated in Table 11.3 is available for the estimates of the parameters of the SRGM [a_i, b_i (in 10^{-4})] and the respective weights of each of the module. For minimizing remaining faults in the system, let the total amount of testing-resource expenditures available be 97,000 in man-hours units. First we determine the allocation of testing resources made for each module according to problem (P1). The results [corresponding to Eqs. (11.2.9) and (11.2.10)] are also tabulated in Table 11.3.

Next using the information of γ_i's as given in Table 11.4 with the same amount of testing resources we again determine the solution following the problem (P4). Let the reliability objective be $R_0 = 0.9$ for the mission time $s = 1.0$ units. The allocation of resources made is tabulated in Table 11.4.

From Tables 11.3 and 11.4 it can be seen that the total software faults remaining are $Z = 98.6$ and 109.1, respectively. It implies that the reduction in fault content according to problem (P1) is 60.7% while according to problem (P4)

Table 11.3 Data and allocated testing resources W_i* according to problem (P1)

Module	a_i	v_i	b_i ($\times 10^{-4}$)	W_i^*	z_i
1	63	1	0.5332	25435.2	16.2
2	13	1	2.5230	5280.7	3.4
3	6	1	5.2620	2459.5	1.6
4	51	1	0.5169	21548.7	16.7
5	15	1	1.7070	6354.5	5.1
6	39	1	0.5723	16554.3	15.1
7	21	1	0.9938	8857.3	8.7
8	9	1	1.7430	3412.3	4.9
9	23	1	0.5057	5845.6	17.1
10	11	1	0.8782	1251.9	9.8

Table 11.4 Data and allocated testing resources W_i^* according to problem (P4)

Module	γ_i (10^{-2})	d_i	W_i^*	z_i
1	0.1800	1,401	23,524.9	16.2
2	1.1240	1,296	4877.0	3.4
3	3.1670	1,121	2265.9	1.6
4	0.2180	1,073	19578.1	16.7
5	0.8560	1,164	5757.8	5.1
6	0.2900	1,214	14,774.5	15.1
7	0.5470	861	7832.3	8.7
8	1.4060	1,050	2827.9	4.9
9	0.4830	1,044	3831.4	17.1
10	1.0850	1,414	1,506.1	9.8

it is 56.5%. The application of (P1) consumes whole of the testing resources, i.e. $W^* = 97{,}000$ man-hours whereas following problem (P4) the total consumption of testing resources is 86,775.90 man-hours. Thus 10,224.10 man-hours are still remaining and we have achieved a reliability level of 0.9. If one further wants to increase the reliability level can decide to continue the testing process or if on the other hand the decision maker wants to release the software before a higher reliability level is achieved, he/she may decide to pace the consumption pattern of testing resources as redundant resources are available. Thus allocation problem with a reliability aspiration constraint provides more flexibility to the decision maker in controlling their testing process.

11.2.6 Minimizing Testing Resources Utilization with a Reliability Objective

Huang et al. [6] reformulated the problem (P2) with a reliability objective, in order to ensure certain minimum level of reliability in each module. The problem is reformulated as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} W_i = W \\ \text{Subject to} \quad & Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i W_i} \\ & W_i \geq 0 \quad i = 1, 2, \ldots N \end{aligned} \tag{P5}$$

$$R = \frac{m_i(t)}{a_i} = 1 - \mathrm{e}^{-b_i W_i(t)} \geq R_0 \tag{11.2.21}$$

Equations (11.2.21) put a reliability aspiration constraint on each module with aspiration level R_0. Again from this we can see that

$$W_i \geq \frac{1}{b_i}\ln(1+R_0), \quad i=1,2,\ldots N \tag{11.2.22}$$

Let $D_i \equiv \frac{1}{b_i}\ln(1+R_0)$, $i=1,2,\ldots N$, now let $X_i = W_i - C_i$, where $C_i = \max(0, D_1, D_2, D_3, \ldots, D_N)$ we can transform problem (P5) to

$$\begin{aligned} &\text{Minimize} \quad \sum_{i=1}^{N}(X_i + C_i) \\ &\text{Subject to} \quad Z = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i C_i}\mathrm{e}^{-b_i X_i} \\ &\qquad\qquad X_i \geq 0 \end{aligned} \tag{P5.1}$$

The objective and first constraints of problem (P5.1) are combined to form the following Lagrange

$$\text{Minimize} \quad L(X_1,X_2,\ldots,X_N,\lambda) = \sum_{i=1}^{N}(X_i+C_i) + \lambda\left(\left(v_i a_i \mathrm{e}^{-b_i C_i}\mathrm{e}^{-b_i X_i}\right) - Z\right) \tag{P5.2}$$

Based on the Kuhn–Tucker (KT) conditions, the necessary conditions for a minimum are

$$\begin{aligned} &\mathbf{A1}: \frac{\partial L(X_1,X_2,\ldots,X_N,\lambda)}{\partial X_1} = 0, \quad i=1,2,\ldots N \\ &\mathbf{A2}: \frac{\partial L(X_1,X_2,\ldots,X_N,\lambda)}{\partial \lambda} = 0, \quad \lambda \geq 0 \\ &\mathbf{A3}: \sum_{i=1}^{N} X_i \leq W - \sum_{i=1}^{N} C_i, W_i \geq 0, \quad i=1,2,\ldots N \end{aligned}$$

From the Kuhn–Tucker conditions we have

$$\frac{\partial L(X_1,X_2,\ldots,X_N,\lambda)}{\partial X_i} = -\lambda v_i a_i b_i \mathrm{e}^{-b_i C_i}\mathrm{e}^{-b_i X_i} + 1 = 0, \quad i=1,\ldots,N \tag{11.2.23}$$

$$\frac{\partial L(X_1,X_2,\ldots,X_N,\lambda)}{\partial \lambda} = \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i C_i}\mathrm{e}^{-b_i X_i} - Z = 0 \tag{11.2.24}$$

and the solution X_i^0 is

$$X_i^0 = \ln\left(\lambda_0 v_i a_i b_i \mathrm{e}^{-b_i C_i}\right)/b_i, \quad i=1,2,\ldots,N \tag{11.2.25}$$

with λ^0

$$\lambda^0 = \left[\frac{\sum_{i=1}^{N}(1/b_i)}{Z}\right] \tag{11.2.26}$$

That is,

$$X_i^0 = \frac{\left(\ln\left(\frac{v_i a_i b_i}{Z} e^{-b_i C_i} \sum_{i=1}^{N} \frac{1}{b_i}\right)\right)}{b_i}, \quad i = 1, 2, \ldots, N \tag{11.2.27}$$

Hence, we get $X^0 = (X_1^0, X_2^0, X_3^0, \ldots, X_N^0)$ as an optimal solution of the Lagrangian problem. However, the above X^0 may have some negative components if

$$v_i a_i b_i e^{-b_i C_i} < \frac{Z}{\sum_{i=1}^{N} \frac{1}{b_i}}$$

making X^0 infeasible for problem (P5.1). In this case, the solution X^0 can be corrected by the following steps.

Algorithm 11.3

1. Set $l = 0$.
2. Calculate

$$X_i = \frac{1}{b_i}\left(\ln\left(\frac{v_i a_i b_i}{Z} e^{-b_i C_i} \sum_{i=1}^{N-l} \frac{1}{b_i}\right)\right), \quad i = 1, 2, \ldots, N - l.$$

3. Rearrange the index i such that

$$X_1^* \geq X_2^* \geq \cdots \geq X_{N-l}^*.$$

4. If $X_{N-l}^* \geq 0$ then stop
   Else update $X_{N-1}^* = 0$; $l = l + 1$
   End-IF.
5. Go to Step 2. The optimal solution has the following form

$$X_i^* = \frac{1}{b_i}\left(\ln\left(\frac{v_i a_i b_i}{Z} e^{-b_i C_i} \sum_{i=1}^{N-l} \frac{1}{b_i}\right)\right), \quad i = 1, 2, \ldots, N - l.$$

Algorithm 11.3 always converges in, at worst, $N - 1$ steps. From $X_i^* \geq 0$ we can determine the optimal allocations as

$$W_i^* = X_i^* + C_i.$$

Application 11.3

Consider the values of a_i, b_i given in Table 11.1 for a software consisting of ten modules obtained from exponential test effort based model (Eq. (11.2.1)). Suppose that the total amount of testing effort expenditures W is 50,000 man-hours and $R_0 = 0.9$. If we consider the weights v_i's [6] as tabulated in Table 11.5 then the

Table 11.5 Resource allocation results of Application 11.3

Module	v_i	W_i^*
1	1.0	6,962
2	0.6	2,608
3	0.7	3,302
4	0.4	3,109
5	1.0	6,258
6	0.2	0
7	0.5	2,847
8	0.6	5,263
9	0.1	0
10	0.5	0

optimal allocation of resources based on problem (P5) following Algorithm 11.3 is as tabulated in Table 11.5.

11.2.7 Minimize the Cost of Testing Resources

The problem (P5) minimizes the testing-resources utilization when the faults remaining in each software module and the minimum reliability level to achieve for each module are given. Huang et al. [7] formulated another type of resource allocation problem where instead of testing-resource consumption the cost of testing resources is minimized. The problem considered is

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} Cost_i(W_i), \\ \text{Subject to} \quad & \sum_{i=1}^{N} W_i \le W \\ & W_i \ge 0 \quad i = 1, 2, \ldots N \\ & R = \frac{m_i(t)}{a_i} = 1 - \mathrm{e}^{-b_i W_i(t)} \ge R_0 \end{aligned} \tag{P6}$$

where the cost function $Cost_i(W_i)$ is defined as the cost of correcting faults during testing and operational phase and the per unit testing expenditure cost in testing phase. Mathematically

$$C(W(t)) = C_1' m(t) + C_2'(m(\infty) - m(t)) + C_3' W(t) \tag{11.2.28}$$

If $m(t)$ is the mean value function of the NHPP and is described by Goel and Okumoto [8] SRGM for each module then

$$Cost_i(W_i(t)) = C_1' m_i(t) + C_2'(m_i(\infty) - m_i(t)) + C_3' W_i(t) \tag{11.2.29}$$

where the cost function $Cost_i(W_i)$ is the cost required to test module i with testing resources W_i, C_1' is the cost of correcting faults in the testing phase, $C_2'$is the fault correction cost in the operational phase and C_3' is the per unit testing expenditure cost for each software module. As described in Sect. 11.2.1 the planning horizon is fixed, therefore, without any loss of generality the number of faults removed by time t can be assumed to be a function of testing effort explicitly in the above equation. Hence the cost function can be expanded as

$$Cost_i(W_i) = C_1' v_i a_i \left(1 - \mathrm{e}^{-b_i W_i}\right) + C_2' v_i a_i \mathrm{e}^{-b_i W_i} + C_3' W_i.$$

Again using the transformations as in (11.2.22) and problem (P5.1) the problem (P6) is reformulated as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} Cost_i(X_i), \\ \text{Subject to} \quad & \sum_{i=1}^{N} X_i \le W - \sum_{i=1}^{N} C_i \\ & X_i \ge 0 \quad i = 1, \ldots, N \end{aligned} \tag{P6.1}$$

where

$$\begin{aligned} \sum_{i=1}^{N} Cost_i(X_i) = C_1' \sum_{i=1}^{N} v_i a_i \left(1 - \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i}\right) + C_2' \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i} \\ + C_3'(X_i + C_i). \end{aligned}$$

Here we assume that the cost function is differentiable [9]. Using the Lagrange multiplier method, the above equation can be reformulated

Minimize

$$\begin{aligned} L(X_1, X_2, \ldots, X_N, \lambda) = C_1' \sum_{i=1}^{N} v_i a_i \left(1 - \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i}\right) \\ + C_2' \sum_{i=1}^{N} v_i a_i \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i} \\ + C_3'(X_i + C_i) + \lambda \left(\sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i \right) \end{aligned} \tag{P6.2}$$

Based on the Kuhn–Tucker conditions, the necessary conditions for obtaining the minimum above are

$$\mathbf{A1}: \frac{\partial L(X_1, X_2, \ldots, X_N, \lambda)}{\partial X_1} = 0, \quad i = 1, 2, \ldots N$$

$$\mathbf{A2}: X_i \frac{\partial L(X_1, X_2, \ldots, X_N, \lambda)}{X_i} = 0, \quad i = 1, 2, \ldots N$$

$$\mathbf{A3}: \lambda\left\{\sum_{i=1}^{N} X_i - \left(W - \sum_{i=1}^{N} C_i\right)\right\} = 0, \quad i = 1, 2, \ldots N.$$

From A1 to A3 we have the following theorem.

Theorem 11.1 *A feasible solution X_i, $i = 1, 2, \ldots, N$ of problem (P6.2) is optimal if and only if*

1. $\lambda \geq v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i} - C_3'$
2. $X_i\left\{\lambda + C_3' - v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i X_i} \mathrm{e}^{-b_i C_i}\right\} = 0.$

From KT conditions we get

$$X_i^0 = \frac{\left(\ln\left(v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-\mathrm{b_i C_i}}\right) - \ln\left(\lambda^0 + C_3'\right)\right)}{b_i}, \quad i = 1, \ldots, N$$

and

$$\lambda^0 = -C_3' + e^{\left(\frac{\left(\sum_{i=1}^{N} (1/b_i) \ln\left(v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i C_i}\right) - W + \sum_{i=1}^{N} C_i\right)}{\sum_{i=1}^{N} 1/b_i}\right)}.$$

Hence, we get $X^0 = (X_1^0, X_2^0, X_3^0, \ldots, X_N^0)$ as an optimal solution of the problem (P6.2). However, the above X^0 may have some negative components if

$$v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i C_i} < \lambda^0 + C_3'$$

making X^0 infeasible for problem (P6.1). In this case, the solution X^0 can be corrected by the following steps.

Algorithm 11.4

1. Set $l = 0$.
2. Calculate

$$X_i = \frac{1}{b_i}\left(\ln\left(v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i C_i}\right) - \ln\left(\lambda + C_3'\right)\right) \quad i = 1, \ldots, N - l$$

$$\lambda = -C_3' + \mathrm{e}^{\left(\frac{1}{\sum_{i=1}^{N} 1/b_i}\right)\left(\left(\sum_{i=1}^{N} (1/b_i) \ln\left(v_i a_i b_i \left(C_2' - C_1'\right) \mathrm{e}^{-b_i C_i}\right) - W + \sum_{i=1}^{N} C_i\right)\right)}.$$

3. Rearrange the index i such that

$$X_1^* \geq X_2^* \geq \ldots \geq X_{N-l}^*.$$

4. IF $X^*_{N-l} \geq 0$ then stop
   Else update $X^*_{N-1} = 0; \quad l = l + 1.$
   End-IF
5. Go to Step 2. The optimal solution has the following form:

$$X^*_i = \frac{1}{b_i}\left(\ln\left(v_i a_i b_i \left(C'_2 - C'_1\right)\mathrm{e}^{-b_i C_i}\right) - \ln\left(\lambda + C'_3\right)\right) \quad i = 1, \ldots, N - l,$$
$$X^*_i = 0, \quad \text{otherwise}$$

where

$$\lambda = -C'_3 + \mathrm{e}^{\left(\frac{1}{\sum_{i=1}^{N} 1/b_i}\right)\left(\left(\sum_{i=1}^{N}(1/b_i)\ln\left(v_i a_i b_i\left(C'_2 - C'_1\right)\mathrm{e}^{-b_i C_i}\right) - W + \sum_{i=1}^{N} C_i\right)\right)}.$$

Algorithm 11.4 always converges in, at worst, $N - 1$ steps. From $X^*_i \geq 0$ we can determine the optimal allocations as

$$W^*_i = X^*_i + C_i.$$

Application 11.4

Again continue with the same modular software data (Table 11.1) of software consisting of ten modules. We need to allocate the expenditures to each module and minimize the expected cost of software during module testing. Let the cost parameters $C'_1 = 2$, $C'_2 = 10$, $C'_3 = 0.5$ and the weighting vector v_i's be as specified in Table 11.6. If we assume we have the total testing effort expenditures (W) amount of 50,000 man-hours and $R_0 = 0.9$, then based on problem (P6) and following Algorithm 11.4 the optimal testing resources are determined and tabulated in Table 11.6.

It is noted that the weight of module 9 is 0.05 (very low) and is not assigned any resources for testing and thus remains untested. From these results we can

Table 11.6 Resource allocation results of Application 11.3

Module	v_i	W^*_i
1	1.0	7,632
2	0.6	3,158
3	0.7	4,009
4	0.4	4,329
5	1.5	8,964
6	0.5	4,568
7	0.5	6,032
8	0.6	9,112
9	0.05	0
10	1	2,203

determine the total expected software testing cost. If for some reasons and specific requirements we intend to decrease more software cost, we have to re-plan and re-consider the allocation of testing-resource expenditures, i.e., with the same data optimal testing-effort expenditures should be re-estimated.

11.2.8 A Resource Allocation Problem to Maximize Operational Reliability

Testing personal, CPU time, test cases, etc., all together are considered as testing resources. The allocation problems discussed in the previous section do not mention what exactly is referred to as testing resources or how it is measured. One can say the total cost of obtaining these resources is testing resource, or the total CPU time available to test the software is the testing resource etc. Xie and Yang [19] studied the allocation problem as testing time allocation problem from the viewpoint of maximizing operational reliability of modular software. To formulate the allocation problem consider the following assumptions along with assumptions 1–4 of Sect. 11.2.1.

5. The total amount of testing time available for the module testing process of software is specified, which is denoted by T.
6. The management has to allocate the total testing time of T to each software module testing process in such a way that the operational reliability of the software system is maximized.

According to Assumption 4, after T_i unit of testing time the failure intensity of software module i is $\lambda_i(T_i)$. Thus, the operational reliability of software module i can be defined as

$$R_i(x) = \mathrm{e}^{-\lambda_i(T_i).x}, \quad x \geq 0 \tag{11.2.30}$$

Authors claimed that, when module i is released after T_i unit of testing time, the latent faults will not be detected and removed. The times between failures in the operational phase will follow an exponential distribution with parameter $\lambda_i(T_i)$, which leads to the formulation of Eq. (11.2.30). The usual equation of reliability [Eq. (1.5.33)] indicates the testing reliability, which describes the reliability growth during the testing phase when faults are removed after they are detected. Operational reliability is oriented toward the customers; hence the formulation (11.2.30) is more appropriate for this problem.

(11.2.30) describes the reliability of a module, from Assumption 3 the operational reliability of the software system is

$$R(x) = \prod_{i=1}^{N} R_i(x) = \mathrm{e}^{-x\sum_{i=1}^{N}\lambda_i(T_i)}, \quad x \geq 0 \tag{11.2.31}$$

maximizing $R(x)$ expressed as (11.2.31) is equivalent to minimizing $\sum_{i=1}^{N} \lambda_i(T_i)$, the optimal testing time allocation problem is formulated as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{N} \lambda_i(T_i) \\ \text{Subject to} \quad & \sum_{i=1}^{N} T_i \le T \\ & T_i \ge 0, \quad i = 1, 2, \ldots, N \end{aligned} \tag{P5}$$

To solve this optimization problem, the following Lagrange is constructed

$$L = \sum_{i=1}^{N} \lambda_i(T_i) + \lambda \left(\sum_{i=1}^{N} T_i - T \right) \tag{11.2.32}$$

The necessary and sufficient conditions for the minimum are

$$\frac{\partial L}{\partial T_i} = \frac{\partial}{\partial T_i} \lambda_i(T_i) + \lambda \ge 0, \quad i = 1, 2, \ldots, N \tag{11.2.33}$$

$$T_i \frac{\partial L}{\partial T_i} = 0, \quad \sum_{i=1}^{N} T_i = T, \quad T_i \ge 0, \quad i = 1, 2, \ldots, N \tag{11.2.34}$$

The optimal solution T_1^*, T_2^*, …, T_n^* can be obtained by solving the above equations. The general formulation presented above does not require a particular model for the mean value function. The author considered a software system where failure process for each software module is described by the Goel and Okumoto [8] model, i.e.

$$\lambda_i(t) = a_i b_i \mathrm{e}^{-b_i t}, \quad i = 1, 2, \ldots, N.$$

The Lagrange becomes

$$L = \sum_{i=1}^{N} a_i b_i \mathrm{e}^{-b_i t} + \lambda \left(\sum_{i=1}^{N} T_i - T \right), \tag{11.2.35}$$

and it can be shown that if modules are indexed in the descending order of $a_i b_i^2$, $i = 1,\ldots, N$ and $a_k b_k^2 > \lambda \ge a_{k+1} b_{k+1}^2$ for any $0 \le k \le N$, then

$$T_i^* = \begin{cases} \frac{1}{b_i}\left(\ln a_i b_i^2 - \ln \lambda\right) & i = 1, 2, \ldots, k \\ 0 & i = k+1, \ldots, N \end{cases} \tag{11.2.36}$$

where

$$\lambda = \mathrm{e}^{\left(\frac{\sum_{i=1}^{k} \frac{1}{b_i} \ln[a_i b_i^2] - T}{\sum_{i=1}^{k} \frac{1}{b_i}} \right)} \tag{11.2.37}$$

Table 11.7 Allocated testing time according to problem (P5)

Module	T_i^*
1	1087.87
2	7139.78
3	8104.65
4	10594.79
5	11457.15
6	11104.05
7	11198.97
8	10233.75
9	4260.38
10	118.59

The algorithm to obtain T_i^*, $i = 1, 2, \ldots, N$.

Algorithm 11.5

1. Set $k = 1$.
2. Calculate the value of the right-hand side of Eq. (11.2.37) and denote it by λ_k.
3. If $a_k b_k^2 > \lambda_k \geq a_{k+1} b_{k+1}^2$, then $\lambda = \lambda_k$ and go to (4); otherwise, set $k = k + 1$ and go back to (2).
4. The optimal solution $T_1^*, T_2^*, \ldots, T_n^*$ can be obtained by Eq. (11.2.36).

Application 11.5

Here we continue with Application 11.1. Consider the same software with a set of ten modules. The estimated figures for a_i's and b_i's are taken from Table 11.1. Suppose 85,000 units of testing time are to be allocated between the ten modules then the application of Algorithm 11.5 yields the allocation of testing time as listed in Table 11.7. Under the optimal allocation, after the testing phase the operational reliability of the software system is

$$R(x) = \exp(-0.01149x), \quad x \geq 0 \tag{11.2.38}$$

However, if the management allocates the testing resource to each software module in proportion to the number of remaining faults in it, then the operational reliability of the software system after testing will be

$$R(x) = \exp(-0.01403x), \quad x \geq 0 \tag{11.2.39}$$

From Eqs. (11.2.38) and (11.2.39) it can be seen that the reliability of the software system is significantly improved by the optimal allocation, compared with that under the other allocation.

11.3 Allocation of Resources for Flexible SRGM

Different types of allocation problems have been discussed in the previous section on the exponential test effort based NHPP SRGM. Throughout the book we have

discussed ample number of NHPP-based SRGM. Most of them are classified as either exponential or s-shaped SRGM. Along with this we have another type of SRGM called flexible SRGM, the shape parameter of these SRGM for different values describes either exponential or s-shaped SRGM. The earlier study in the software reliability growth modeling as well as resource allocation problem was focused on exponential SRGM. Development of s-shaped and flexible SRGM and their wide range application in practice invoked the requirement of studying this optimization problem on these SRGM as well. Kapur et al. [10] initiated this study and firstly they proposed and validated a test effort based flexible SRGM, and then formulated resource allocation optimization problems on this model.

11.3.1 Maximizing Fault Removal During Testing Under Resource Constraint

Under the general NHPP assumptions (refer Sect. 2.3.1) and assuming

- The fault detection rate with respect to testing effort intensity is proportional to the current fault content in the software and the proportionality increases linearly with each additional fault removal.
- Faults present in the software are of two types: mutually independent and mutually dependent.

The model is formulated as

$$\frac{(d/dt)m(t)}{w(t)} = \phi(t)(a - m(t)) \quad \text{where } \phi(t) = b\left[r + (1-r)\frac{m(t)}{a}\right] \tag{11.3.1}$$

where r is called the inflection parameter and represents the proportion of independent faults present in the software. Other notations have their usual meanings. The mean value function of the SRGM under the initial condition $m(0) = 0$ and $W(0) = 0$ is

$$m(t) = \frac{a\left(1 - \mathrm{e}^{-bW(t)}\right)}{1 + ((1-r)/r)\mathrm{e}^{-bW(t)}} \tag{11.3.2}$$

Depending upon the value of r, the SRGM (108) can describe both exponential and S-shaped growth curves. The behavior of the testing effort can be described by any of the testing effort functions discussed in Sect. 2.7.

The Problem Formulation

From the estimates of parameters of SRGM for software modules, the total fault content in the software $\sum_{i=1}^{N} a_i$ is known. Module testing aims at detecting maximum number of faults within available resources. The SRGM with testing effort for ith module is given as

$$m_i(t) = \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i(t)}\right)}{1 + ((1 - r_i)/r)_i \mathrm{e}^{-b_i W_i(t)}}, \quad i = 1, 2, \ldots, N \tag{11.3.3}$$

Again, it is not imperative that software will be tested indefinitely to detect/remove possible fault content due to the random life cycle of the software, which has to be released for marketing. Hence, the software-testing time is almost fixed (say T). Let W_i be the testing effort that has to be spent on the ith module during testing time T, so the mean value function of SRGM can be rewritten explicitly as a function of W_i

$$m_i(W_i) = \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i}\right)}{1 + ((1 - r_i)/r_i)\mathrm{e}^{-b_i W_i}}, \quad i = 1, 2, \ldots, N \tag{11.3.4}$$

With the mean value function (11.3.4) to describe the fault detection process the allocation problem with the objective of maximum fault removal during testing subject to the resource availability constraint is formulated as

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N} m_i(W_i) = \sum_{i=1}^{N} \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i}\right)}{1 + ((1 - r_i)/r_i)\mathrm{e}^{-b_i W_i}} \\ \text{Subject to} \quad & \sum_{i=1}^{N} W_i \le W \\ & W_i \ge 0, \quad i = 1, 2, \ldots, N \end{aligned} \tag{P7}$$

(P7) can be solved using the Dynamic Programming approach. From Bellman's principle of optimality, we can write the following recursive equations [11]

$$f_1(W) = \max_{W_1 = W} \left\{ \frac{a_i\left(1 - \mathrm{e}^{-b_1 W_1}\right)}{1 + ((1 - r_1)/r_1)\mathrm{e}^{-b_1 W_1}} \right\} \tag{11.3.5}$$

$$f_n(W) = \max_{0 \le W_n \le W} \left\{ \frac{a_n\left(1 - \mathrm{e}^{-b_n W_n}\right)}{1 + ((1 - r_n)/r_n)\mathrm{e}^{-b_n W_n}} + f_{n-1}(W - W_n) \right\}, \quad n = 2, \ldots, N \tag{11.3.6}$$

Let

$$p_i(W_i) = a_i(1 - \mathrm{e}^{-b_i W_i}), \quad q_i(W_i) = 1 + \delta_i \mathrm{e}^{-b_i W_i}$$

and

$$R_i(W_i) = (p_i(W_i)/q_i(W_i)), \quad i = 1, \ldots, N.$$

where

$$\delta_i = \frac{(1 - r_i)}{r_i}, \quad i = 1, \ldots, N.$$

The derivatives of $p_i(W_i)$ and $q_i(W_i)$ are ever non-increasing and non-decreasing functions of W_i, respectively. The functions $p_i(W_i)$ and $q_i(W_i)$, $i = 1,\ldots, N$, are hence concave and convex, respectively. The ratio of concave and convex functions is a pseudo-concave function and the sum of pseudo-concave functions is not necessarily a pseudo-concave function. There does not exist any direct method to obtain an optimal solution for such a class of problems. Dur et al. [12] proposed a method to solve such a class of problems converting the sum of ratio functions of the objective to a multiple objective fractional programming problem. Further, it has been established that every optimal solution of the original problem is an efficient solution of the equivalent multiple objective fractional programming problem. Dur's equivalent of the problem (P7) can be written as

$$\begin{aligned} &\text{Maximize} \quad R(W) = (p_1(W_1)/q_1(W_1), \quad p_2(W_2)/q_2(W_2), \ldots, p_N(W_N)/q_N(W_N))^T \\ &\text{Subject to} \quad W \in S = \left\{ W \in R^N / \sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0, \quad i = 1, 2, \ldots, N \right\} \end{aligned} \tag{P7.1}$$

Problem (P7.1) can equivalently be written as the following multiple objective programming problem [13]

$$\begin{aligned} &\text{Maximize} \quad R(W) = (p_1(W_1) - q_1(W_1), \ p_2(W_2) - q_2(W_2), \ldots, p_N(W_N) - q_N(W_N))^T \\ &\text{Subject to} \quad W \in S = \left\{ W \in R^N / \sum_{i=1}^{N} W_i \le W, \ W_i \ge 0, \ i = 1, 2, \ldots, N \right\} \end{aligned} \tag{P7.2}$$

The Geoffrion's equivalent scalarized formulation [14] with suitable adjustment (i.e., taking both functions together having the same variable) of the problem (P7.2) for fixed weights for the objective function is as follows

$$\begin{aligned} &\text{Maximize} \quad \sum_{i=1}^{N} \lambda_i (p_i(W_i) - q_i(W_i)) \\ &\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W \\ &\qquad W_i \ge 0, \quad i = 1, 2, \ldots, N \\ &\qquad \lambda \in \Omega = (\lambda \in R^N / \sum \lambda = 1_i, \quad \lambda_i \ge 0, \quad i = 1, \ldots, N) \end{aligned} \tag{P7.3}$$

Based on the following Lemma it can be proved that the optimal solution (W_i^*, $i = 1,\ldots, N$) of the problem (P7.3) is an optimal solution (W_i^* for $i = 1,\ldots, N$) for the problem (P7).

Lemma 1 [12]: *The optimal solution X* of the problem (P7) is an efficient solution of the problem (P7.2).*

Lemma 2 [13]: *A properly efficient solution (W_i^* for i=1,…, N) of the problem (P7.3) is also a properly efficient solution (W_i^* for i = 1,…, N) for the problem (P7.1).*

Lemma 3 [14]: *The optimal solution (W_i^* for i = 1,…, N) of the problem (P7.3) is a properly efficient solution (W_i^* for i = 1,…, N) for the problem (P7.2).*

Now, the problem (P7.3) can be solved using the Dynamic Programming Approach. The recursion equations can be written after substituting the expressions for $p_i(W_i)$ and $q_i(W_i)$, $i = 1,\ldots, N$ and simplifying

$$f_1(W) = \max_{W_1 = W} \left\{(a_1 - 1) - (a_1 + \delta_1)\mathrm{e}^{-b_1 W_1}\right\} \tag{11.3.7}$$

$$f_n(W) = \max_{0 \le W_n \le W} \left\{(a_n - 1) - (a_n + \delta_n)\mathrm{e}^{-b_n W_n} + f_{n-1}(W - W_n)\right\}, \quad n = 2, \ldots, N \tag{11.3.8}$$

The modules can be rearranged in decreasing order of their values of $(a_i + \delta_i)b_i$; i.e., $(a_1 + \delta_1)b_1 \ge (a_2 + \delta_2)b_2 \ge \cdots \ge (a_N + \delta_N)b_N$ to index them. The resources are allocated sequentially to modules starting from the module having higher detectability, determined by $(a_i + \delta_i)b_i$ to module having low detectability. The above problem can be solved through forward recursion in N stages as follows

Stage 1: Let $n = 1$, then we have

$$f_1(W) = \max_{X_1 = Z} \left\{(a_1 - 1) - (a_1 + \delta_1)\mathrm{e}^{-b_1 W_1}\right\} = \left\{(a_1 - 1) - (a_1 + \delta_1)\mathrm{e}^{-b_1 W}\right\}$$

Stage 2: Set $n = 2$, then we have

$$f_2(W) = \max_{0 \le X_2 \le Z} \left\{(a_2 - 1) - (a_2 + \delta_2)\mathrm{e}^{-b_2 W_2} + f_1(W - W_2)\right\}$$

Substitute for $f_1(W - W_2)$, and let $f_2(W) = \max_{0 \le X_2 \le Z}\{F_2(W_2)\}$. The function $F_2(W_2)$ can be maximized using the principles of calculus. Proceed by induction to find the optimal solution for the nth stage. The result can be summarized in the following theorem. For detailed proof see Kapur et al. [10].

Theorem 11.2 *If for any $n = 2, \ldots, N$, $1 \le \mathrm{e}^{-\mu_{n-1}} W \le (\mu_{n-1} V_{n-1}/(a_n + \delta_n)b_n)$ then the values of $W_n, W_{n+1}, \ldots, W_N$ are zero and the problem reduces to an $(n - 1)$ stage problem with*

$$W_r^* = \frac{1}{b_r + \mu_{r-1}}\left[\mu_{r-1} W - \log\left(\frac{\mu_{r-1} V_{r-1}}{(a_r + \delta_r)b_r}\right)\right], \quad r = 1, \ldots, (n-1) \tag{11.3.9}$$

where

$$\mu_r = \frac{1}{\sum_{j=1}^{i}(1/b_j)} \tag{11.3.10}$$

and

$$\mu_i V_i = \prod_{j=1}^{i} \left((a_j + \delta_j) b_j\right)^{(\mu_i / b_j)}, \quad i = 1, \ldots, N \tag{11.3.11}$$

The objective function value and modulewise faults removed corresponding to the optimal allocation of testing resource (W_i^* for $i = 1,\ldots, N$) are given as

$$f_{n-1}(W) = \sum_{i=1}^{n-1} (a_i - 1) - V_{n-1} \mathrm{e}^{-\mu_{n-1} W} \tag{11.3.12}$$

$$m_i(W_i^*) = \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i^*}\right)}{1 + \delta_i \mathrm{e}^{-b_i W_i^*}}, \quad i = 1, \ldots, (n-1) \tag{11.3.13}$$

where n may take values varying from 2 to N. In this allocation procedure, some of the modules may not get any resources. The management may not agree to such a situation where one or more modules are not tested. It is always desired during module testing that each of the modules is adequately tested so that a certain minimum reliability level is achieved for the software as well as for each of the modules. In other words, a certain percentage of the fault content is desired to be removed in each module of the software. Hence, allocation problem (P7) needs to be suitably modified to maximize the removal of faults in the software under resource and the minimum desired level of faults to be removed from each of the modules in the software constraints. The resulting testing resource allocation problem can be stated as follows

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N} m_i(W_i) = \sum_{i=1}^{N} \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i}\right)}{1 + \delta_i \mathrm{e}^{-b_i W_i}} \\
\text{Subject to} \quad & m_i(W_i) = \frac{a_i\left(1 - \mathrm{e}^{-b_i W_i}\right)}{1 + \delta_i \mathrm{e}^{-b_i W_i}} \geq p_i a_i = a_{i0}, \quad i = 1, \ldots, N \\
& \sum_{i=1}^{N} W_i \leq W \\
& W_i \geq 0, \quad i = 1, 2, \ldots, N
\end{aligned} \tag{P8}$$

where a_{i0}, $i = 1,\ldots, N$ is the minimum number of faults that must be detected from each of the software modules.

From constraint

$$\frac{a_i\left(1 - \mathrm{e}^{-b_i W_i}\right)}{1 + \delta_i \mathrm{e}^{-b_i W_i}} \geq a_{i0}, \quad i = 1, \ldots, N \tag{11.3.14}$$

we get

$$W_i \geq -\frac{1}{b_i}\log\left[\frac{1-(a_{i0}/a_i)}{1+(a_{i0}/a_i)\delta_i}\right] = Z_i \text{ (say)}, \quad i = 1,\ldots,N \tag{11.3.15}$$

Let

$$Y_i = W_i - Z_i, \quad i = 1,\ldots,N \tag{11.3.16}$$

Therefore, (P8) through the problem (P7)–(P7.3) can be restated as

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N} \lambda_i\left\{(\bar{a}_i - 1) - (\bar{a}_i + \delta_i)\mathrm{e}^{-b_i Y_i}\right\} \\ \text{Subject to} \quad & \sum_{i=1}^{N} Y_i \leq W - \sum_{i=1}^{N} Z_i = \overline{Z} \ \text{ (say)} \\ & Y_i \geq 0, \quad i = 1,\ldots,N \\ & \bar{a}_i = a_i - a_{i0}, \quad i = 1,\ldots,N \end{aligned} \tag{P8.1}$$

Problem (P8.1) is similar to Problem (P7.3) and, hence, using Theorem 11.2 Problem (P7) can also be solved in the same manner. The result is summarized in Theorem 11.3.

Theorem 11.3 *If for any* $n = 2,\ldots,N$, $1 \leq \mathrm{e}^{-\mu_{n-1}\bar{Z}} \leq (\mu_{n-1}\bar{V}_{n-1}/(\bar{a}_n + \delta_n)b_n)$, *then values of* Y_n, Y_{n+1}, …, Y_N *are zero and the problem reduces to an* $(n-1)$ *stage problem with*

$$Y_r^* = \frac{1}{b_r + \mu_{r-1}}\left[\mu_{r-1}\overline{Z} - \log\left(\frac{\mu_{r-1}\overline{V}_{r-1}}{(\bar{a}_r + \delta_r)b_r}\right)\right], \quad r = 1,\ldots,(n-1) \tag{11.3.17}$$

where $\mu_i = 1/\left(\sum_{j=1}^{i}(1/b_j)\right)$

$$\mu_i\overline{V}_i = \prod_{j=1}^{i}\left((\bar{a}_j + \delta_j)b_j\right)\left(\mu_i/b_j\right), \quad i = 1,\ldots,(n-1)$$

and the corresponding objective function value is

$$f_{n-1}(\bar{Z}) = \sum_{i=1}^{n-1}(\bar{a}_i - 1) - \bar{V}_{n-1}\mathrm{e}^{-\mu_{n-1}\bar{Z}}.$$

The total number of faults removed from each of the modules is given as

$$m_i(W_i^*) = m_i(Z_i + Y_i^*), \quad i = 1,\ldots,N \tag{11.3.18}$$

Application 11.6

Consider software having eight modules testing. It is assumed that the parameters a_i, b_i and r_i for the ith module $i = 1,\ldots, 8$ have already been estimated using the failure data. The hypothetical parameter values are listed in Table 11.8. Suppose the total resource available for testing is 110,000 units. Problem (P7) is solved

Table 11.8 Data and results of Application 11.6 problem (P7)

Module	a_i	b_i	r_i	W_i^*	m_i^*	Faults removed (%)	Faults remaining (%)
1	45	0.00041	0.85412	05658.82	40	89	11
2	13	0.00032	0.88524	05279.45	10	80	20
3	16	0.00026	0.88959	06416.74	13	80	20
4	35	0.00015	0.78999	11922.74	28	80	20
5	14	0.00009	0.79578	13086.36	9	64	36
6	21	0.00006	0.75492	19814.15	13	62	38
7	20	0.00003	0.58921	27712.49	9	46	54
8	11	3.15115	0.57863	20109.24	4	34	66
Total	175			110,000	126	72	28

Table 11.9 Results of Application 11.6 problem (P8) with aspiration 50%

Module	a_i	a_{i0}	Z_i^*	Y_i^*	m_i (Y_i)	W_i^*	m_i^*	Faults removed (%)	Faults remaining (%)
1	45	23	1935.7	2053.59	10	3989.30	33	73	27
2	13	7	2627.2	587.40	1	3214.60	8	62	38
3	16	8	2851.3	687.87	1	3539.15	9	56	44
4	35	18	5664.3	2573.72	5	8238.05	23	66	34
5	14	7	9086.4	0	0	9086.36	7	50	50
6	21	11	15306	0	0	15306.5	11	52	48
7	20	10	30991	0	0	30990.6	10	50	50
8	11	6	35636	0	0	35635.5	6	55	45
Total	175	90	104097	5902.58	16	110,000	107	61	39

using the recursion Eqs. (11.3.7) and (11.3.8) and optimal allocations of resources (W_r^*) for the modules are computed from Eq. (11.3.9). The results are listed in Table 11.8 along with the corresponding expected number of faults removed calculated through Eq. (11.3.13), percentages of faults removed and faults remaining for each module.

The total number of faults that can be removed through this allocation is 126 (i.e., 72% of the total fault content is removed). It is observed that in some modules, the number of faults remaining after allocation is higher than the removed faults. This can lead to frequent failure during the operational phase. Obviously, this will not satisfy the developer and he may desire that at least 50% of the fault content from each of the modules of the software is removed (i.e., $p_i = 0.5$ for each $i = 1,\ldots, 8$). Since faults in each module are integral values, the nearest integer larger than 50% of the fault content in each module is taken as the lower limit that has to be removed. The new allocation of resources along with the expected number of faults removed, percentages of faults removed, and faults remaining for each module after solving Problem (P8) computed through Eqs. (11.3.17) and (11.3.18) is summarized in Table 11.9. The total number of faults that can be removed through this allocation is 107 (i.e., 61% of

Table 11.10 Results of Application 11.6 problem (P8) with aspiration 70%

Module	a_i	a_{i0}	Z_i^*	Y_i^*	$m_i(Y_i)$	W_i^*	m_i^*	Faults removed (%)	Faults remaining (%)
1	45	23	1935.7	3286.34	16	5222.04	39	86	14
2	13	7	2627.2	2149.9	3	4777.1	10	77	23
3	16	8	2851.3	2751.95	4	5603.23	12	74	26
4	35	18	5664.3	5389.16	8	11,053.5	26	74	26
5	14	7	9086.4	2325.25	1	11,411.6	8	57	43
6	21	11	15,306	0	0	15,306.5	11	52	48
7	20	10	30,991	0	0	30,990.6	10	50	50
8	11	6	35,636	0	0	35,635.5	6	55	45
Total	175	90	104,097	15902.6	31	120,000	122	70	30

the total fault content is removed from the software). In addition to the above, if it is desired that a certain percentage of the total fault content is to be removed, then additional testing resources would be required.

It is interesting to study this tradeoff and Table 11.10 summarizes the corresponding results, where the required percentage of faults removed is 70%. To achieve this, 10,000 units of additional testing resource are required. The total number of faults that can be removed through this allocation is 122 (i.e., 70% of the fault content is removed from the software). The analysis given in Tables 11.8, 11.9 and 11.10 helps in providing the developer to have an insight into resource allocation and the corresponding fault removal phenomenon, and the objective can be set accordingly.

11.3.2 Minimizing Testing Cost Under Resource and Reliability Constraint

The results obtained from the allocation problem of maximizing the fault removal during testing may suggest consuming all of the available testing resources. On the other hand the decision maker might be interested in allocating the available resources such that the cost incurred can be minimized, i.e. some of the available resources can be saved simultaneously satisfying the reliability requirements. In Sect. 11.2 we have discussed one such problem on an exponential test effort based model. In this section we will discuss the formulation and solution methodology of an allocation problem minimizing testing cost under resource and reliability constraints in which the fault detection process is described by the flexible SRGM specified by Eq. (11.3.4) [15].

Problem Formulation

For formulating the allocation problem first we must model the cost function for the testing phase and the debugging cost incurred during the operational phase.

Consider the cost function (11.2.19), as we consider the modules to be independent of each other so is their testing process. In modular software some of the modules are small and have simple coding, some others are large and complex and few can be of medium size and/or complexity. The fault removal cost for each software module is hence usually different. To consider this consideration the cost function (11.3.29) for the ith module can be modified as

$$Cost_i(W_i(t)) = C_{1i}m_i(t) + C_{2i}(m_i(\infty) - m_i(t)) + C_3'W_i(t) \tag{11.3.19}$$

or

$$Cost_i(W_i(t)) = C_{1i}m_i(t) + C_{2i}(a_i - m_i(t)) + C_3'W_i(t) \tag{11.3.20}$$

Using cost function (11.3.20) and SRGM (11.3.4), with upper bound of available resources W and reliability objective R_0 the problem is formulated as

$$\text{Minimize} \quad C(W) = \sum_{i=1}^{N} C_i(W_i) = \sum_{i=1}^{N} (C_{1i} - C_{2i})v_i \frac{f_i(W_i)}{g_i(W_i)} + \sum_{i=1}^{N} C_{2i}a_i + C_3' \sum_{i=1}^{N} W_i$$

$$\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W \quad i = 1, \ldots, N$$

$$W_i \ge 0 \quad i = 1, \ldots, N$$

$$R_i(t) = \frac{1 - e^{-b_i W_i}}{1 + \delta_i e^{-b_i W_i}} \ge R_0 \tag{P9}$$

where $f_i(W_i) = a_i(1 - e^{-b_i W_i})$, $g_i(W_i) = 1 + \delta_i e^{-b_i W_i}$. The derivatives of $f_i(W_i)$ and $g_i(W_i)$ are ever non-increasing and non-decreasing functions of W_i, respectively, therefore the functions $f_i(W_i)$ and $g_i(W_i)$, $i = 1,\ldots, N$ are, respectively concave and convex. The ratio of concave and convex functions is pseudo-concave function and the sum of pseudo-concave functions is not necessarily a pseudo-concave function. The second term of the cost function is a constant and hence can be dropped and the third term is a linear function of W_i. Dropping the constant term from the objective function and rewriting the above problem as the maximization problem, the equivalent problem can be restated in terms of expected gain as follows

$$\text{Maximize} \quad G(W) = -\sum_{i=1}^{N} C_i(W_i) = \sum_{i=1}^{N} (C_{2i} - C_{1i})v_i \frac{f_i(W_i)}{g_i(W_i)} - C_3' \sum_{i=1}^{N} W_i$$

$$\text{Subject to} \quad \sum_{i=1}^{N} W_i \le W, \quad i = 1, \ldots, N$$

$$W_i \ge 0, \quad i = 1, \ldots, N$$

$$R_i(t) = \frac{1 - e^{-b_i W_i}}{1 + \delta_i e^{-b_i W_i}} \ge R_0 \tag{P9.1}$$

Such a problem cannot be solved directly to obtain an optimal solution. Dur et al. [12] transformation of objective function to a multiple objective fractional programming problem is

$$\text{Maximize}\quad G(W)=\left(\gamma_1\frac{f_1(W_1)}{g_1(W_1)},\ldots,\gamma_N\frac{f_N(W_N)}{g_N(W_N)},-C_3'\sum_{i=1}^{N}W_i\right)$$

$$\text{Subject to}\quad W\in S=\left\{W\in R^N\Big/\sum_{i=1}^{N}W_i\le W,\quad W_i\ge 0,\quad R_i(t)\ge R_0,\quad i=1,\ldots,N\right\} \tag{P9.2}$$

where $\gamma_i = (C_{2i} - C_{1i})v_i$. Further the problem (P9.1) can equivalently be written as the following multiple objective programming problem [13]

$$\text{Maximize}\quad G(W)=\left(\gamma_1(f_1(W_1)-g_1(W_1)),\ldots,\gamma_N(f_N(W_N)-g_N(W_N)),-C_3'\sum_{i=1}^{N}W_i\right)$$

$$\text{Subject to}\quad W\in S=\left\{W\in R^N\Big/\sum_{i=1}^{N}W_i\le W, W_i\ge 0, R_i(t)\ge R_0,\ \ i=1,\ldots,N\right\} \tag{P9.3}$$

The Geoffrion's [14] scalarized formulation with suitable adjustment (i.e. taking both functions together having same variable) of the problem (P9.3) for fixed weights for the objective function is

$$\begin{aligned}
\text{Maximize}\quad & \sum_{i=1}^{N}\lambda_i\gamma_i(f_i(W_i)-g_i(W_i))-\lambda_{N+1}C_3'\sum_{i=1}^{N}W_i\\
\text{Subject to}\quad & \sum_{i=1}^{N}W_i\le W\quad i=1,\ldots,N\\
& W_i\ge 0\quad i=1,\ldots,N\\
& R_i(t)=\frac{1-e^{-b_iW_i}}{1+\delta_i e^{-b_iW_i}}\ge R_0\\
& \lambda\in\Omega=\left(\lambda\in R^{N+1}/\sum\lambda_i=1,\ \lambda_i\ge 0;\ i=1,\ldots,N+1\right)
\end{aligned} \tag{P9.4}$$

Lemmas 1–3 of Sect. 11.3.1 applies to problem (P9.1) to (P9.4). Based on the Lemma the following theorem is derived.

Theorem 11.4 *If equal relative importance is attached to each of the objectives of the problem (P9.4) [i.e. $\lambda_i = 1/(N + 1)$ for $i = 1,\ldots, N$] or simply we can take $\lambda_i = 1$ for $i = 1,\ldots, N$ for problem (P9.4) and (W_i^* for $i = 1,\ldots, N$) is an optimal solution of the problem (P9.5) then (W_i^* for $i = 1,\ldots, N$) is also an optimal solution for the problem (P9.1) and hence problem (P9).*

From Theorem 11.4 it remains to find the optimal solution of problem (P9.4) assuming $\lambda_i = 1$ for $i = 1,\ldots, N$ to find the optimal solution of problem (P9).

Using the constraint on reliability objective for *i*th module the problem (P9.4) is transformed as follows

$$W_i \geq \frac{-1}{b_i} \ln\left(\frac{1-R_0}{1+R_0\delta_i}\right) \equiv C_i, \quad i = 1, \ldots, N \text{ (say)} \tag{11.3.20}$$

Let $X_i = \mathrm{W_i} - C_i$, the problem (P9.4) can be rewritten as

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N} \gamma_i (f_i(X_i + C_i) - g_i(X_i + C_i)) - C_3' \sum_{i=1}^{N} (X_i + C_i) \\ \text{Subject to} \quad & \sum_{i=1}^{N} X_i \leq W - \sum_{i=1}^{N} C_i, \quad i = 1, \ldots, N \\ & X_i \geq 0, \quad i = 1, \ldots, N \end{aligned} \tag{P9.5}$$

Further substituting the values of $f_i(X_i + C_i)$, $g_i(X_i + C_i)$ and γ_i in (P9.5), it can be restated as

$$\begin{aligned} \text{Maximize} \quad & \left(\begin{array}{l} \displaystyle\sum_{i=1}^{N} (C_{2i} - C_{1i}) v_i \left(a_i \left(1 - \mathrm{e}^{-b_i(X_i+C_i)} \right) - \left(1 + \delta_i \mathrm{e}^{-b_i(X_i+C_i)} \right) \right) \\ \\ \displaystyle - C_3' \sum_{i=1}^{N} (X_i + C_i) \end{array} \right) \\ \text{Subject to} \quad & \sum_{i=1}^{N} X_i \leq W - \sum_{i=1}^{N} C_i, \quad i = 1, \ldots, N \\ & X_i \geq 0 \quad i = 1, \ldots, N \end{aligned} \tag{P9.6}$$

In the problem (P9.6) the functions $f_i(X_i) = a_i\left(1 - \mathrm{e}^{-b_i(X_i+C_i)}\right)$ and $g_i(X_i) = \left(1 + \delta_i \mathrm{e}^{-b_i(X_i+C_i)}\right)$, $i = 1, \ldots, N$ are concave and convex, respectively. Negative of a convex function is a concave function, hence $-g_i(W_i) i = 1,\ldots,$ N are concave functions. Functions $X_i + C_i$, $i = 1,\ldots, N$ are linear, hence they may be treated as convex functions. The positive linear combination of concave functions $f_i(W_i)$, $-g_i(W_i)$ and $-(X_i + C_i)$, $i = 1,\ldots, N$ is concave. Hence the objective function is a concave function. The constraint other than non-negative restriction is linear. Hence the above problem is a convex programming problem and the necessary Kuhn–Tucker conditions of optimality for convex programming problem are also sufficient. Following saddle value problem is formulated for problem (P9.6).

$$\begin{aligned} \max_{X_i} \min_{\theta} \phi(X_1, X_2, \ldots X_N, \theta) \\ = \sum_{i=1}^{N} (C_{2i} - C_{1i}) v_i \left(a_i \left(1 - \mathrm{e}^{-b_i(X_i+C_i)} \right) - \left(1 + \delta_i \mathrm{e}^{-b_i(X_i+C_i)} \right) \right) \\ - C_3' \sum_{i=1}^{N} (X_i + C_i) + \theta \left(\sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i \right) \end{aligned} \tag{P9.7}$$

The saddle point of saddle value problem (P9.7) provides an optimal solution to the problem (P9.5) and hence optimal for problems (P9.4) and (P9). The necessary and sufficient conditions for (X^*, θ^*), where $X^* = \{X_i : i = 1,\ldots, N\}$ to be a saddle point for the saddle value problems are based on the KT conditions and are given by the following theorem.

Theorem 11.5 *A feasible solution X_i, $i = 1,\ldots, N$ of problem (P9.7) is optimal if and only if*

1. $\theta \le C_3' - (C_{2i} - C_{1i})v_i b_i \mathrm{e}^{-b_i(X_i+C_i)}(a_i + \delta_i)$.
2. $X_i\{\theta - C_3 + (C_{2i} - C_{1i})v_i b_i \mathrm{e}^{-b_i(X_i+C_i)}(a_i + \delta_i)\} = 0$.

Corollary 11.5 *Let X_i be a feasible solution of problem (P9.6)*

$X_i = 0$ if and only if $\theta \le C_3' - (C_{2i} - C_{1i})v_i b_{ie}^{-b_i C_i}(a_i + \delta_i)$

If $X_i > 0$, then

$$X_i = \{\ln((C_{2i} - C_{1i})v_i b_i \mathrm{e}^{-b_i C_i}(a_i + \delta_i)) - \ln(C_3' - \theta)\}/b_i$$

Finding a Feasible Solution at Optimality Condition

Applying KT conditions to the problem (P9.7) we have

$$\frac{\partial\phi(X_1, X_2, \ldots X_N, \theta)}{\partial X_i} = (C_{2i} - C_{1i})v_i b_i(a_i + \delta_i)\mathrm{e}^{-b_i(X_i+C_i)} - C_3' + \theta = 0,$$
$$i = 1, \ldots, N$$

which implies

$$X_i^0 = \{\ln((C_{2i} - C_{1i})v_i b_i \mathrm{e}^{-b_i C_i}(a_i + \delta_i)) - \ln(C_3' - \theta)\}/b_i, \quad i = 1, \ldots, N \tag{11.3.21}$$

and

$$\frac{\partial\phi(X_1, X_2, \ldots X_N, \theta)}{\partial\theta} = \sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i = 0$$

which implies

$$\theta^0 = C_3' - \exp\left[\frac{\sum_{i=1}^{N}(1/b_i)\ln((C_{2i} - C_{1i})v_i b_i \mathrm{e}^{-b_i C_i}(a_i + \delta_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N}(1/b_i)}\right] \tag{11.3.22}$$

$X^0 = (X_1^0, X_2^0, \ldots, X_N^0)$ can have some negative components if $(C_{2i} - C_{1i})v_i b_i e^{-b_i C_i}(a_i + \delta_i) < \theta + C_3'$, which will make X^0 infeasible for problem (P9.6). If the above

case arises, then the solution of X^0 can be corrected to obtain a feasible solution by the following algorithm.

Algorithm 11.6

1. Set $S = 0$.
2. Calculate X_i, $i = 1,\ldots, N - S$; θ using Eqs. (11.3.21) and (11.3.22)

$$X_i = \frac{1}{b_i}\left\{\ln\left((C_{2i} - C_{1i})v_i b_i e^{-b_i C_i}(a_i + \delta_i)\right) - \ln\left(C_3' - \theta\right)\right\}, \quad i = 1,\ldots,N - S$$

$$\theta = C_3' - \exp\left[\frac{\sum_{i=1}^{N-S}(1/b_i)\ln\left((C_{2i} - C_{1i})v_i b_i e^{-b_i C_i}(a_i + \delta_i)\right) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N-S}(1/b_i)}\right]$$

3. Rearrange index i in the ascending order of allocation

$$X_1 \geq X_2 \geq X_3 \geq \cdots \geq X_{N-S}.$$

4. If $X_{N-S} \geq 0$ then Stop (the solution is optimal)
   Else $X_{N-S} = 0$; set $S = S + 1$
   End if.
5. For re-allocating testing resources to remaining N–S modules go to Step 2**.**
   The optimal solution is given by

$$X_i^* = \frac{1}{b_i}\left\{\ln\left((C_{2i} - C_{1i})v_i b_i e^{-b_i C_i}(a_i + \delta_i)\right) - \ln\left(C_3' - \theta\right)\right\}, \quad i = 1,\ldots,N - l$$

$$X_i^* = 0, \text{ otherwise,}$$

where

$$\theta = C_3' - \exp\left[\frac{\sum_{i=1}^{N-S}(1/b_i)\ln\left((C_{2i} - C_{1i})v_i b_i e^{-b_i C_i}(a_i + \delta_i)\right) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N-S}(1/b_i)}\right].$$

Algorithm 11.6 converges in, at worst, $(N - 1)$ steps.
The value of objective function at the optimal solution $\left(X_1^*, X_2^*, \ldots, X_N^*\right)$ is

$$C(X^*) = \sum_{i=1}^{N}(C_{1i} - C_{2i})v_i\left(a_i\left(1 - e^{-b_i(X_i^* + C_i)}\right) - \left(1 + \delta_i e^{-b_i(X_i^* + C_i)}\right)\right) + C_3'\sum_{i=1}^{N}\left(X_i^* + C_i\right).$$

Now $W_i^* = X_i^* + C_i$, $i = 1,\ldots, N$ is an optimal solution of problem (P9.4), which in turn is an optimal solution of problems (P9.1) and (P9).

Table 11.11 Data and allocation results of Application 11.7

Module	a_i	b_i	δ_i	v_i	W_i	Cost
1	22	0.001413	0.85412	0.6	3048.687	1640.405
2	16	0.001032	0.885236	0.7	3710.713	1927.949
3	12	0.000642	0.889588	0.4	3908.611	2039.378
4	20	0.000501	0.789985	1.5	8176.666	4053.359
5	11	0.000109	0.795781	0.5	23040.11	11590.46
6	14	0.000315	0.754922	0.6	8115.211	4137.126
				Total	50,000	25388.67

Application 11.7
Consider software having six modules that are being tested during module testing. It is assumed that parameters a_i, b_i, r_i and v_i for each of the six modules have already been estimated using the failure data and are tabulated in Table 11.11. The total testing resource available is assumed to be 50,000 units. It is desired that each software module reaches a reliability level of at least 0.9. Assume that the cost of correcting a fault in testing and operational phase is respectively same for each of the module and the cost parameters are $C_{i1} = C'_1 = 2$, $C_{i2} = C'_2 = 10$, $i = 1,\ldots 6$ and $C'_3 = 0.5$ units. Using the above information the problem is solved following Algorithm 11.6 and the testing effort allocated to each of the modules and the total expected cost of each of the modules is also listed in Table 11.11.

The calculated amount of total testing effort allocated is $W^* = 50{,}000$ and the total minimum expected cost of testing all the modules such that reliability of each of the modules is at least 0.9 is equal to 25388.67 units.

11.4 Optimal Testing Resource Allocation for Test Coverage Based Imperfect Debugging SRGM

Testing coverage is an important aspect of software testing. Allocating resources during the module according to an optimization problem in which the testing process is represented by a testing coverage based SRGM can some times be more accurate and favored by the decision makers. Jha et al. [16] proposed a test coverage measure based imperfect debugging SRGM and formulated an optimization problem based on this model. Inclusion of imperfect debugging phenomenon brings the results more closely to the real testing process. Based on the general assumptions of NHPP and further assuming

1. No new faults are introduced in the software system during the testing phase.
2. The failure process is dependent upon testing coverage.
3. The fault removal at any time during testing is a function of the number of failures with a time lag.

The following differential equation is formed

$$\frac{m_f'(t)}{w(t)} = \frac{q'}{c-q}\left[a - m_f(t)\right] \tag{11.4.1}$$

here q' is the rate (with respect to testing effort) with which the software is covered through testing and c is the proportion of total software which will be eventually covered during the testing phase, $0 < c < 1$. If c is closer to 1 one can conclude that test cases were efficiently chosen to cover the operational profile. For a logistic fault removal rate we can assume the following form of $q'/(c - q)$

$$q = q(W(t)) = c\frac{1 - \mathrm{e}^{bpW(t)}}{1 + \beta\mathrm{e}^{-bpW(t)}} \tag{11.4.2}$$

Equation (11.4.2) directly relates testing effort to testing coverage, because with more testing effort we can expect to cover more portion of the software. Here p is the probability of perfect debugging $0 \le p \le 1$. The mean value function of the SRGM with respect to (11.4.1) using (11.4.2) with initial conditions $m_f(0) = 0$ and $W(0) = 0$ is

$$m_f(t) = a\frac{1 - \mathrm{e}^{-bpW(t)}}{1 + \beta\mathrm{e}^{-bpW(t)}} \tag{11.4.3}$$

On a failure observation, attempts are made to remove the cause of the failure. The removal process is dependent upon the number of failures at any time instant. But there is a definite time lag between the two processes. Hence the fault removal process can be represented by the following equation

$$m_r(W) = m_f(W - \Delta W) \tag{11.4.4}$$

ΔW is the additional effort during the removal time lag. Different time-dependent forms of the lag function can be considered depending upon the testing environment. As the number of faults reduces and the chance of checking the same path for faults increases it also results in increase in time lag. Hence we assume an increasing form of Δt [17] as

$$\Delta W = \frac{1}{bp}\ln(1 + bpW) \tag{11.4.5}$$

Eq. (11.4.5) implies

$$m_r(t) = a\frac{1 - (1 + bpW(t))\mathrm{e}^{-bpW(t)}}{1 + \beta(1 + bpW(t))\mathrm{e}^{-bpW(t)}} \tag{11.4.6}$$

Authors have validated the model on several data sets.

11.4.1 Problem Formulation

The optimization problem for maximizing the number of faults that can be removed during the testing process under the budget constraint and lower bounds on the number of fault removals from each of the software removal is formulated here. The lower bounds assure a minimum level of reliability that can be achieved by the testing resource allocation. Solution methodology of any such optimization requires that all the problem coefficients must be known a priori. The constant coefficients involved in the problem are either estimated from the past failure history or by experience. Therefore assuming that the values of all the coefficients are known, the product of the coefficients b and p (a constant) is replaced by a single constant b.

The problem for finding the optimal amount of testing resource to be allocated to module i, which would maximize the removal of total faults, is formulated by defining the mean value function of SRGM explicitly as a function of testing resources. The reason for this as already been stated in the previous sections.

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N} m_i(W_i) = \sum_{i=1}^{N} \frac{a_i\left(1-(1+b_iW_i)\mathrm{e}^{-b_iW_i}\right)}{1+\beta_i(1+b_iW_i)\mathrm{e}^{-b_iW_i}} \\
\text{Subject to} \quad & m_i(W_i) = \frac{a_i\left(1-(1+b_iW_i)\,\mathrm{e}^{-b_iW_i}\right)}{1+\beta_i(1+b_iW_i)\mathrm{e}^{-b_iW_i}} \geq N_{i0}, \quad i=1,\ldots,N \\
& \sum_{i=1}^{k} W_i \leq Z, \quad i=1,\ldots,N \\
& W_i \geq 0, \quad i=1,\ldots,N
\end{aligned} \tag{P10}$$

Here N_{i0} is the aspired minimum number of faults to be removed from each of the modules. As in problems (P8) and (P9) let $f_i(W_i) = a_i\left(1-(1+b_iW_i)\mathrm{e}^{-b_iW_i}\right)$, $g_i(W_i) = 1+\beta_i(1+b_iW_i)\mathrm{e}^{-b_iW_i}$ and $F_i(W_i) = f_i(W_i)/g_i(W_i) i = 1,\ldots, N$. Hence resulting problem (P10) becomes maximization of a sum of ratios (fractional functions) under specified testing-resource expenditure and the minimum level of the removal of faults of each module which is again a fraction and can be written as follows

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N} F_i(W_i) \\
\text{Subject to} \quad & F_i(W_i) \geq N_{i0} \quad i=1,\ldots,N \\
& \sum_{i=1}^{N} W_i \leq Z \\
& W_i \geq 0 \quad i=1,\ldots,N
\end{aligned} \tag{P10.1}$$

The derivatives of $f_i(W_i)$ and $g_i(W_i)$, $i = 1, \ldots, N$ are non-increasing and non-decreasing functions of W_i, respectively, hence the functions $f_i(W_i)$ and $g_i(W_i)$

$i = 1, \ldots, N$ are concave and convex, respectively. The ratio of concave and convex functions is pseudo-concave function and the sum of pseudo-concave functions is not necessarily a pseudo-concave function and due to non-existence of any direct method to obtain an optimal solution for such class of problems we state the Dur et al. [12] formulation of the problem (P10.1).

$$\begin{aligned} &\text{Maximize} \quad F(W) = (F_1(W_1)\ldots F_K(W_K))^T \\ &\text{Subject to} \quad W \in S \end{aligned} \tag{P10.2}$$

where $S = \left\{ W \in R^k / F_i(W_i) \geq N_{i0}, \sum_{i=1}^{k} W_i \leq Z, \text{ and } W_i \geq 0, \quad i = 1, \ldots, N \right\}$.

Although the problem (P10.2) can be solved using dynamic programming approach as applied in Sect. 11.3, here the goal programming approach is chosen for finding the solution primarily to underscore the importance of tradeoff between testing effort (that contributes to cost) and number of faults detected in each module. Through goal programming approach the aspirations of the management can be controlled. To formulate problem (P10.2) as goal programming problem, the following concepts of multi-objective programming (i.e. definitions and lemmas) have been used.

11.4.2 Finding Properly Efficient Solution

Definition 1 [20] : A function $f_i(W_i)$ $i = 1, \ldots, N$ is said to be pseudo concave if for any two feasible points $F_i(W_{i1}) \geq F_i(W_{i2})$ implies $F_i'(W_{i1})(W_{i2} - W_{i1}) \leq 0$.

Definition 2 [21] : A feasible solution $W^* \in S$ is said to be an efficient solution for the problem (P10.3) if there exists no $W \in S$ such that $F(W) \geq F(W^*)$ and $F(W) \neq F(W^*)$.

Definition 3 [21]: An efficient solution $W^* \in S$ is said to be a properly efficient solution for the problem (P10.2) if there exists $\alpha > 0$ such that for each r, $(F_r(W) - F_r(W^*))/\left(F_j(W^*) - F_j(W)\right) < \alpha$ for some j with $F_j(W) < F_j(W^*)$ and $F_r(W) > F_r(W^*)$ for $W \in S$.

Let $y_i = F_i(W_i) = [f_i(W_i)/g_i(W_i)]$ $i = 1, \ldots, k$, then the equivalent parametric problem for multiple objective fractional programming problem (P10.2) is given as

$$\begin{aligned}
&\text{Maximize} \quad y = (y_1, \ldots, y_k)^T \\
&\text{Subject to} \quad f_i(W_i) - y_i g_i(W_i) \geq 0 \quad i = 1, \ldots, N \\
&\qquad\qquad y_i \geq N_{i0} \quad i = 1, \ldots, N \\
&\qquad\qquad \sum_{i=1}^{N} W_i \leq Z \\
&\qquad\qquad W_i \geq 0 \quad i = 1, \ldots, N
\end{aligned} \tag{P10.3}$$

The Geoffrion's [14] scalarization for the problem (P10.3) for fixed weights of the objective functions is as follows

$$\begin{aligned}
&\text{Maximize} \quad \sum_{i=1}^{N} \lambda_i y_i \\
&\text{Subject to} \quad f_i(W_i) - y_i g_i(W_i) \geq 0 \quad i = 1, \ldots, N \\
&\qquad\qquad y_i \geq N_{i0} \quad i = 1, \ldots, N \\
&\qquad\qquad \sum_{i=1}^{N} W_i \leq Z \\
&\qquad\qquad W_i \geq 0 \quad i = 1, \ldots, N \\
&\qquad\qquad \lambda \in \Omega = \left(\lambda \in R^k / \sum \lambda_i = 1,\ \lambda_i \geq 0;\ i = 1, \ldots, k\right)
\end{aligned} \tag{P10.4}$$

Based on Lemma 1, Lemma 2 and Lemma 3 it can be proved that an optimal solution of the problem (P10.4) taking $\lambda_i = 1$ for $i = 1, \ldots, N$ is also an optimal solution of the original problem (P10). It remains to obtain an optimal solution of the problem (P10.4) taking $\lambda_i = 1$ for $i = 1, \ldots, N$. The problem (P10.4) can be solved by standard mathematical programming approach using any NLP solver like LINGO software if there exists a feasible solution. The problem results into an infeasible solution if either minimum level of fault removal in some or each of the modules of the software is very high or management is interested in setting target for total fault removal for the software. In such situation standard mathematical programming approach may not provide a solution and goal programming approach (GPA) [18] can be useful.

11.4.3 Solution Based on Goal Programming Approach

In a simpler version of goal programming, management sets goals and relative importance (weights) for different objectives. Then an optimal solution is defined as one that minimizes the both positive and negative deviations from set goals simultaneously or minimizes the amount by which each goal can be violated. Now if management wishes to remove at least $p \sum_{i=1}^{N} a_i$ number of total faults from the software the problem (P10.1) can be rewritten as

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{N} F_i(W_i) \\
\text{Subject to} \quad & F_i(W_i) \geq N_{i0} \quad i = 1, \ldots, N \\
& \sum_{i=1}^{N} W_i \leq Z \\
& W_i \geq 0 \quad i = 1, \ldots, N \\
& \sum_{i=1}^{N} F_i(W_i) \geq p \sum_{i=1}^{N} a_i
\end{aligned} \tag{P10.5}$$

In GPA, we first solve the problem using rigid constraints only and then the goals of objectives are incorporated depending upon whether priorities or relative importance of different objectives is well defined or not. The problem (P10.5) can be solved in two stages as follows

$$\begin{aligned}
\text{Minimize} \quad & g_0(\eta, \rho, W) = \sum_{i=1}^{N} \eta_i + \rho_{N+1} \\
\text{Subject to} \quad & f_i(W_i) - N_{i0} g_i(W_i) + \eta_i - \rho_i = 0 \quad i = 1, \ldots, N \\
& \sum_{i=1}^{N} W_i + \eta_{N+1} - \rho_{N+1} = Z \\
& W_i \geq 0 \quad i = 1, \ldots, N \\
& \eta_i, \rho_i \geq 0 \quad i = 1, \ldots, N+1
\end{aligned} \tag{P10.6}$$

where η_i and ρ_i are over- and underachievement (positive and negative deviational) variables from the goals for the objective/constraint function i, respectively, and $g_0(\eta, \rho, W)$ is the goal objective function corresponding to the rigid constraint functions. The choice of deviational variable in the goal objective functions, which has to be minimized, depends upon the following rule. Let $f(W)$ and b be the function and its goal, respectively, and η_i and ρ_i be the over- and underachievement (positive and negative deviational) variables then

- if $f(W) \leq b$, ρ is minimized under the constraints $f(W) + \eta - \rho = b$
- if $f(W) \geq b$, η is minimized under the constraints $f(W) + \eta - \rho = b$ and
- if $f(W) = b$, $\eta + \rho$ is minimized under the constraints $f(W) + \eta - \rho = b$.

Let (η^0, ρ^0, W^0) be the optimal solution for the problem (P10.6) and $g_0(\eta^0, \rho^0, W^0)$ be its corresponding objective function value then the second stage problem can be formulated using optimal solution of the problem (P10.6) through the problem (P10.5)

Table 11.12 Data and results of Application 11.8

a_i	b_i	p_i	β_i	Allocation with 60% removal from each module		Allocation with 60% removal from each module and total 70%	
				W_i^*	$m_i(W_i^*)$	W_i^*	$m_i(W_i^*)$
1,349	0.0034	0.870	1.58	995.84	812	1120.82	918
1,309	0.0037	0.884	2.13	1015.58	825	1126.44	919
1,356	0.0027	0.902	1.33	1153.27	814	1153.27	814
1,332	0.0049	0.925	20.40	1387.30	1044	1442.09	1091

$$
\begin{aligned}
\text{Minimize} \quad & g(\eta, \rho, W) = \sum_{i=N+2}^{2N-1} \eta_i + \rho_{2N} \\
\text{Subject to} \quad & f_i(W_i) - y_i g_i(W_i) + \eta_{N+1+i} - \rho_{N+1+i} = 0 \quad i = 1, \ldots, N \\
& y_i + \eta_i - \rho_i = a_{i0} \quad i = 1, \ldots, N \\
& \sum_{i=1}^{N} W_i + \eta_{k+1} - \rho_{k+1} = Z \\
& \sum_{i=1}^{N} y_i + \eta_{2N+2} - \rho_{2N+2} = p \sum_{i=1}^{N} a_{i0} \\
& W_i \geq 0 \quad i = 1, \ldots, N \\
& \eta_i, \rho_i \geq 0 \quad i = 1, \ldots, 2N+2 \\
& g_0(\eta, \rho, W) = g_0\left(\eta^0, \rho^0, W^0\right)
\end{aligned}
\tag{P10.7}
$$

where $g(\eta, \rho, W)$ is the objective function of GPA corresponding to the objective. The problem (P10.7) may be solved by standard mathematical programming approach and numerical solution can be obtained using optimization software such as LINGO.

Application 11.8

Consider a software that consists of four modules and the values of the parameters a_i, b_i, β_i and p_i of the fault removal SRGM (11.4.6) for the ith software module ($i = 1,\ldots,4$) as listed in Table 11.12. Let the total testing effort resource available be 4,552 units and it is targeted that at least 60% of the total faults should be removed during testing from each of the modules. With these data, the problem (P10.3) is solved and the results are given in columns 5 and 6 of Table 11.12. Now if the management fixes a target of removing at least 60% of the faults from each of the modules and 70% of the total faults, the resulting problem has no feasible solution within the available resources. Goal programming approach is used here to obtain a compromise solution. Columns 7 and 8 of Table 11.12 gives optimal testing resources allocated (W_i^*, $i = 1\ldots4$) and the corresponding number of faults removed ($m_i(W_i^*)$, $i = 1,\ldots,4$) for each of the software modules.

From the above table we can compute that to have a minimum aspiration of 60% minimum fault removal from each software module following problem (P10), total of 4,552 units of testing resources are required, i.e. all of the resources get consumed to remove a total of 3,495 faults from the total 5,346. It implies the 65.38% faults can be removed from this allocation. So now if the management aims to remove a minimum 70% of the faults from the software then 4842.62 units of resources are required, i.e. 290.62 extra units of resources are required. The

Module	a_i	b_i ($\times 10^{-4}$)	v_i
1	89	4.1823	1.0
2	25	5.0923	1.5
3	27	3.9611	1.3
4	45	2.2956	0.5
5	39	2.5336	2.0
6	39	1.7246	0.3
7	59	0.8819	1.7
8	68	0.7274	1.3
9	37	0.6824	1.0
10	14	1.5309	1.0

fault removal count would now increase to 3,742 from 3,495.

Exercises

1. What is an allocation problem of testing resources?
2. Assume that the weights assigned to the modules in Application 11.1 are changed according to the weighting vector (0.12, 0.08, 0.08, 0.13, 0.11, 0.08, 0.08, 0.14, 0.14, 0.04), What will be the change in the resource allocation if problem is solved according to Problem P1.
3. Consider software with ten modules. The information tabulated in the following table is available for the estimates of the parameters of the exponential SRGM $m_i(t) = a_i\left(1 - \mathrm{e}^{-b_i t}\right)$ $\left(a_i,\, b_i \text{ (in } 10^{-4})\right)$ and the respective weights of each of the modules. If the total amount of testing-resource expenditures available is 50,000 in man-hours units. Determine the allocation of testing resources for each module
4. Suppose after computing the optimal testing effort allocations in Application 11.4 it is found that the estimate of initial fault content of module 1 is wrongly noted as 89 faults instead of actual 125. Due to this we need to determine the optimum allocations again, using Algorithm 11.4. Determine the correct allocations. Does the allocation of all the modules change due to this change if yes, give the correct solution?
5. The cost of correcting faults during testing and operational phases is taken to be same for all modules in Application 11.7. It may not be true. Assume that the cost of correcting faults in operational phase for module 1 is 20 units while it

remains 10 units for all other modules. How the optimal allocation of testing resources would change due to this change in cost for module 1.

References

1. Ohtera H, Yamada S (1990) Optimal allocation and control problems for software testing resources. IEEE Trans Reliab 39(2):171–176
2. Kubat P, Koch HS (1983) Managing test-procedure to achieve reliable software. IEEE Trans Reliab 32(3):299–303
3. Yamada S, Hishitani J, Osaki S (1991) Test-effort dependent software reliability measurement. Int J Syst Sci 22(1):73–83
4. Yamada S, Ichimori T, Nishiwaki M (1995) Optimal allocation policies for testing-resource based on a software reliability growth model. Math Comput Model 22:295–301
5. Leung YW (1997) Dynamic resource allocation for software-module testing. J Syst Softw 37(2):129–139
6. Huang CY, Lo JH, Kuo SY, Lyu MR (2002) Optimal allocation of testing resources for modular software systems. In: Proceedings of 13th IEEE International Symposium on Software Reliability Engineering (ISSRE 2002), Nov 2002, Annapolis, Maryland, pp 129–138
7. Huang CY, Lo JH, Kuo SY, Lyu MR (2004) Optimal allocation of testing-resource considering cost, reliability, and testing-effort. In: Proceedings of 10th IEEE/IFIP Pacific Rim International Symposium on Dependable Computing, Papeete, Tahiti, French Polynesia, pp 103–112
8. Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Trans Reliab R 28(3):206–211
9. Kubat P (1989) Assessing reliability of modular software. Oper Res Lett 8(1):35–41
10. Kapur PK, Jha PC, Bardhan AK (2004) Optimal allocation of testing resource for a modular software. Asia Pacific J Oper Res 21(3):333–354
11. Hadley G (1964) Nonlinear and dynamic programming. Addison Wesley, Reading, MA
12. Dur M, Horst R, Thoai NV (2001) Solving sum of ratios fractional programs using efficient points. Optimization 1:447–466
13. Bhatia D, Kumar N, Bhudhiraja RK (1997) Duality theorem for non differential multi-objective programs. Indian J Pure Appl Math 28(8):1030–1042
14. Geoffrion AM (1968) Proper efficiency and theory of vector maximization. J Math Anal Appl 22:613–630
15. Jha PC, Gupta D, Yang B, Kapur PK (2009) Optimal testing-resource allocation during module testing considering cost, testing effort and reliability. Comput Ind Eng 57(3):1122–1130
16. Jha PC, Gupta D, Anand S, Kapur PK (2006) An imperfect debugging software reliability growth model using lag function with testing coverage and related allocation of testing effort problem. Commun Dependability Qual Manag: Int J 9(4):148–165
17. Xie M, Zhao M (1992) The Schneidewind software reliability model revisited. In: Proceedings of 3rd International Symposium on Software Reliability Engineering, pp 184–192
18. Ignizio JP (1994) Linear programming in single and multiple objective functions. Prentice Hall, Englewood Cliffs, London
19. Xie M , Yang B (2000) Optimal testing time allocation for modular system, Int J Qual Reliab Manag 11(8):854–863.
20. Bazaraa SM, Sherali HD, Setty CM (1993) Non linear programming: Theory and algorithm, John Wiley and Sons, New York.
21. Steuer RE (1989) Multiple criteria optimization: Theory, computation and application, Wiley, New York

Chapter 12
Fault Tolerant Systems

12.1 Introduction

In the 21st century we seldom see any industry or service organization working without the help of an embedded software system. Such a dependence of mankind on software systems has made it necessary to produce highly reliable software. Complex safety critical systems currently being designed and built are often difficult multi-disciplinary undertakings. Part of these systems is often a computer control system. In order to ensure that these systems perform without failure, even under extreme conditions, it is important to build extremely high reliability in them, both for hardware and software. There are many real life examples when failures in computer systems of safety critical systems have caused spectacular failure resulting in calamitous loss to life and economy. In the recent years hardware systems have attained very high reliability with the introduction of recent technologies and productive design methods. To increase the reliability further the technique of building redundancy is quite favorable. Hardware redundancy techniques simply imply the use of some extra resources in order to tolerate the faults. Redundancy in hardware is usually implemented in static (passive), dynamic (active) or hybrid form. The purpose is that concurrent computation can be voted upon, errors can be masked out, or redundant hardware can be switched automatically to replace failed components.

Means to cope with the existence and manifestation of faults in software are divided into three main categories

- *Fault avoidance/prevention* This includes use of software design methodologies, which attempt to make software provably fault-free.
- *Fault removal* These methods aim to remove faults after the development stage is completed. Exhaustive and rigorous testing of the final product does this.

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_12,
© Springer-Verlag London Limited 2011

- *Fault tolerance* This method makes the assumption that the system has unavoidable and undetectable faults and aims to make provisions for the system to operate correctly even in the presence of faults.

Before we start further discussion on how these methodologies can be implemented to attain very high reliability in software, we tell the readers how important it is to achieve.

Short Summary of History's Worst Ten Software Bugs, Appearance

Software bugs are with us since the times when computing systems came into existence and show no signs of going extinct. As the line between software and hardware blurs, coding errors are increasingly playing tricks on our daily lives. There is a big list of software failures and it is hard to rate their severity. Which is worse—a security vulnerability that is exploited by a computer worm to shut down the internet for a few days or a typo that triggers a day-long crash of the nation's phone system? The answer depends on whether you want to make a phone call or check your e-mail.

July 28, 1962—Mariner I Space Probe

A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean. The investigation into the accident discovers that a formula written on paper with pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory.

1982—Soviet Gas Pipeline

Operatives working for the Central Intelligence Agency allegedly (.pdf) plant a bug in a Canadian computer system purchased to control the trans-Siberian gas pipeline. The Soviets had obtained the system as part of a wide-ranging effort to covertly purchase or steal sensitive US technology. The CIA reportedly found out about the program and decided to make it backfire with equipment that would pass Soviet inspection and then fail once in operation. The resulting event is reportedly the largest non-nuclear explosion in the planet's history.

1985–1987—Therac-25 Medical Accelerator

A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation:

either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.What engineers did not know was that both the Therac-20 and Therac-25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others were seriously injured.

1988—Buffer Overflow in Berkeley Unix Finger Daemon

The first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called *gets*() designed to get a line of text over the network. Unfortunately, *gets*() has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect. Programmers respond by attempting to stamp out the *gets*() function in working code, but they refuse to remove it from the C programming language's standard input/output library, where it remains to this day.

1988–1996—Kerberos Random Number Generator

The authors of the Kerberos security system neglect to properly "seed" the program's random number generator with a truly random seed. As a result, for 8 years it is possible to trivially break into any computer that relies on Kerberos for authentication. It is unknown if this bug was ever actually exploited.

January 15, 1990—AT&T Network Outage

A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines—a message that the neighbors send out when they recover from a crash. One day a switch in New York crashes and reboots, causing its neighboring switches to crash, then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every 6 s, leaving an estimated 60,000 people without long distance service for 9 h. The fix: engineers load the previous software release.

1993—Intel Pentium Floating Point Divide

A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range. For example, dividing 4195835.0/3145727.0 yields 1.33374 instead of 1.33382, an error of 0.006%. Although the bug affects few users, it becomes a public relations nightmare. With an estimated 3–5 million defective chips in circulation, at first Intel only offers to replace Pentium chips for consumers who can prove that they need high accuracy; eventually the company relents and agrees to replace the chips for anyone who complains. The bug ultimately costs Intel $475 million.

1995/1996—The Ping of Death

A lack of sanity checks and error handling in the IP fragmentation reassembly code makes it possible to crash a wide variety of operating systems by sending a malformed "ping" packet from anywhere on the internet. Most obviously affected are computers running Windows, which lock up and display the so-called "blue screen of death" when they receive these packets. But the attack also affects many Macintosh and UNIX systems as well.

June 4, 1996—Ariane 5 Flight 501

Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing. First Flight 501's backup computer crashes, followed 0.05 s later by a crash of the primary computer. As a result of these crashed computers, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 s after launch.

November 2000—National Cancer Institute, Panama City

In a series of accidents, therapy planning software created by Multidata Systems International, a US firm, miscalculates the proper dosage of radiation for patients undergoing radiation therapy.

Multidata's software allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five. The doctors discover that they can trick the software by drawing all five blocks as a single large

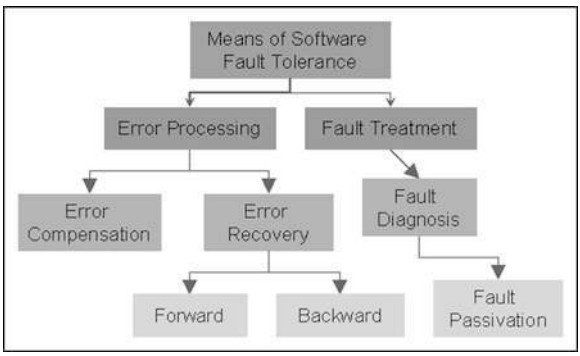


Fig. 12.1 Fault tolerant strategies for software

block with a hole in the middle. What the doctors do not realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure.

At least eight patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder.

It can be understood that all these incidences of software failure would have created a very critical situation when they occurred. Now if we make a statement that after the implementation of techniques of fault avoidance and removal we can assure that no more than 1% of software faults which were present in the software initially are remaining at its release, can we make out the cost and effort needed to make a guarantee against this remaining number. There is only one solution which is, *fault tolerance*, the only remaining hope to achieve dependable software. Fault tolerance makes it possible for the software system to provide service without failure even in the presence of faults. This means that an imminent failure needs to be prevented or recovered from.

At this stage first we must understand the nature of faults. Software faults are either permanent (Bohrbugs) or transient (Heisenbugs). A fault is said to be permanent if it continues to exist until it can be repaired and most of them can be removed through rigorous and extensive testing and debugging. Both fault avoidance and removal methodologies employed to attain high system reliability target mostly on Bohrbugs. A transient software fault is one that occurs and disappears at an unknown frequency. The remaining faults in software after testing and debugging are usually heisenbugs which eluded detection during the testing. So it is mainly heisenbugs that need to be tolerated by the technique of fault tolerance.

12.2 Software Fault Tolerance Techniques

There are mainly two strategies for software fault tolerance—*error processing* and *fault treatment*. Error processing aims to remove errors from the software state and

Table 12.1 Strategies used by different fault tolerance methods

Technique → Strategy↓	Design diversity	Data diversity	Environment diversity
Error compensation	Yes	Yes	–
Error recovery	Yes	–	–
Fault treatment	–	–	Yes

can be implemented by substituting an error-free state in place of the erroneous state, called *error recovery*, or by compensating for the error by providing redundancy, called *error compensation*. Error recovery can be achieved by either forward or backward error recovery. The second strategy is, fault treatment, it aims to prevent activation of faults and so action is taken before the error creeps in. The two steps in this strategy are *fault diagnosis* and *fault passivation*. Figure 12.1 shows this classification of fault tolerance systems. The nature of faults, which typically occur in software, has to be thoroughly understood in order to apply these strategies effectively.

Techniques for tolerating faults in software have been divided into four classes—design diversity, data diversity, checkpoint and recovery and environment diversity. Table 12.1 shows the fault tolerance strategies used by these classes.

Design Diversity

Design diversity techniques are specifically developed to tolerate design faults in software arising out of wrong specifications and incorrect coding. The method requires redundant software elements that provide alternative means to fulfill the same specifications. The aim is to obtain system survival on some input by means of a correct output from at least one of the alternatives hence, no system failure on most occasions. Software reliability engineering technique suggests using different specifications; design, programming languages and team, algorithms to build the alternative versions so that the independent version fails independently with least possible common failures. These variants are used in a time or space redundant manner to achieve fault tolerance. Popular techniques, which are based on the design diversity concept for fault tolerance in software, are—Recovery Block (RB), N-Version Programming (NVP) and N-self-Checking Programming. Few hybrid schemes are also proposed by some researchers [1].

Data Diversity

The technique of data diversity, a technique for fault tolerance in software, was introduced by Ammann and Knight [2]. The approach uses only one version of the software and relies on the observation that a software sometime fails for certain values in the input space and this failure could be averted if there is a minor perturbation of input data which is acceptable to the software. N-copy programming, based on data diversity, has N copies of a program executing in parallel, but

each copy running on a different input set produced by a diverse-data system. The diverse-data system produces a related set of points in the data space. Selection of the system output is done using an enhanced voting scheme which may not be a majority voting mechanism, with minor perturbation of input data. This technique might not be acceptable to all programs since equivalent input data transformations might not be acceptable by the specification. However, in some cases like a real-time control program, a minor perturbation in sensor values may be able to prevent a failure since sensor values are usually noisy and inaccurate. This technique is cheaper to implement than design diversity techniques.

Environment Diversity

Environment diversity is the newest approach to fault tolerance in software. Although this technique has been used for long in an ad hoc manner, only recently has it gained recognition and importance. Having its basis on the observation that most software failures are transient in nature, the environment diversity approach requires re-executing the software in a different environment [3]. Environment diversity deals very effectively with Heisenbugs by exploiting their definition and nature. Adams [4] has proposed that restarting the system is the best approach to masking software faults. Environment diversity is a generalization of restart. Environment diversity attempts to provide a new or modified operating environment for the running software. Usually, this is done at the instance of a failure in the software. When the software fails, it is restarted in a different, error-free operating system environment state, which is achieved by some clean up operations.

Examples of environment diversity techniques include retry operation; restart application and rebooting the node. The retry and restart operations can be done on the same node or on another spare (cold/warm/hot) node. Tandem's fault tolerant computer system [5] is based on the *process pair* approach. It was noted that these failures did not recur once the application was restarted on the second processor. This was due to the fact that the second processor provided a different environment, which did not trigger the same error, conditions which led to the failure of the application on the first processor. Hence in this case, hardware redundancy was used to tolerate most of the software faults.

Among the three fault tolerant techniques design diversity is a concept that traces back to the very early age of informatics [6, 7]. The approach is the most widely used and has become a reality, as witnessed by the real life systems. The currently privileged domain where design diversity is applied is the domain of safety related systems and hence is very important to study and understand. The two most well-documented techniques of design diversity for tolerating software design faults are—the RB and the NVP that we describe in detail in the later sections. Both the schemes are based on the technique of protective software redundancy assuming that the events of coincidental software failures are rare.

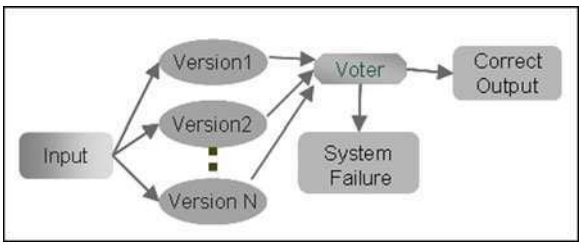


Fig. 12.2 N-version programming scheme

12.2.1 N-*version Programming Scheme*

The NVP scheme came into existence with the work of Chen and Avizienis [8] for the design diversity technique of software fault tolerance. Conceptually the scheme is to independently generate $N > 2$ functionally equivalent programs (modules) called as "*VERSIONS*" for the same initial specification of a given task. This concept is similar to the NMR (N-modular programming) approach in hardware fault tolerance. By independent generation of programs it means that different versions are developed by different individuals or groups, who are independent of each other in the sense that they there is no communication and interactions between them. The different teams use various design diversity techniques such as use of different algorithms, techniques, process models, programming languages, environment and tools in order to obtain the aim of fault tolerance by providing a means to avoid coincidental failures in independent versions. In a NVP system the *N*-program versions for any particular application are executed in parallel on identical input and the results are obtained by voting on the outputs from the individual programs under the assumption that the original specifications provided to the programming teams are not flawed. Voting is performed by a voting mechanism, which is similar in concept to a decision mechanism. It is a voter when more than two versions are installed in parallel and is a comparator in case of a 2VP system. Several voting techniques have been proposed in the literature. The most commonly seen and the simplest one is the majority voting, in this usually N is odd and the voter needs at least $[N/2]$ software versions to produce the same output to determine the majority as the correct output. The other commonly known technique is consensus voting designed for multi-version software with small output space, where software versions can give identical but incorrect outputs. The voter will select the output given by most of the versions. Leung [9] proposed the use of maximum likelihood estimation to decide the most likely correct result for small output spaces. Figure 12.2 shows the implementation of a NVP scheme. Use of a NVP system is expensive, difficult to maintain, and its repair is not trivial.

The probability of failure of the NVP scheme, P_{NVP} can be expressed as

$$P_{\mathrm{NVP}} = \prod_{i=1}^{n} e_i + \prod_{i=1}^{n} (1 - e_i) e_i^{-1} \prod_{j=1}^{n} e_j + d \qquad (12.2.1)$$

Assume all N versions are statistically independent of each other and have the same reliability r, and if majority voting is used, then the reliability of the NVP scheme can be expressed as

$$R_{\mathrm{NVP}} = \sum_{i=\lceil N/2 \rceil}^{N} \binom{N}{i} r^i (1-r)^{N-i} \tag{12.2.2}$$

where e_i is the probability of failure in version i. and d is the probability that there are at least two correct results but the voter fails to deliver the correct result.

12.2.2 Recovery Block Scheme

Recovery blocks were first coined by Horning et al. [10] although they gained popularity after the study of Randell [7]. This scheme is analogous to the cold standby scheme for hardware fault tolerance. The basic recovery block relates to sequential systems. Basically, in this approach, multiple variants of software, which are functionally equivalent, are deployed in a time redundant fashion. On entry to a recovery block the state of the system must be saved to permit backward error recovery, i.e. establish a checkpoint. The primary alternate is executed and then the acceptance test is evaluated to provide adjudication on the outcome of this primary alternate. If the acceptance test is passed then the outcome is regarded as successful and the recovery block can be exited, discarding the information on the state of the system takes non-entry (i.e. checkpoint). However, if the test fails or if any errors are detected by other means during the execution of the alternate, then an exception is raised and backward error recovery is invoked. This restores the state of the system to what it was on entry. After such recovery, the next alternate is executed and then the acceptance test is applied again. This sequence continues until either an acceptance test is passed or all alternates have failed the acceptance test. If all the alternates either fail the test or result in an exception (due to an internal error being detected), a failure exception will be signaled to the environment of the recovery block. Since recovery blocks can be nested, then the raising of such an exception from an inner recovery block would invoke recovery in the enclosing block. Figure 12.3 shows the implementation of a recovery block and the operation of the recovery block can be illustrated by Fig. 12.4.

The probability of failure of the recovery block scheme, P_{RB}, is defined as

$$P_{\mathrm{RB}} = \prod_{i=1}^{n} (e_i + t_{2i}) + \sum_{i=1}^{n} t_{1i} e_i \left(\prod_{j=1}^{i-1} (e_j + t_{2j}) \right) \tag{12.2.3}$$

where t_{1i} is the probability that acceptance test i judges an incorrect result as correct t_{2i} is the probability that acceptance test i judges an correct result as incorrect.

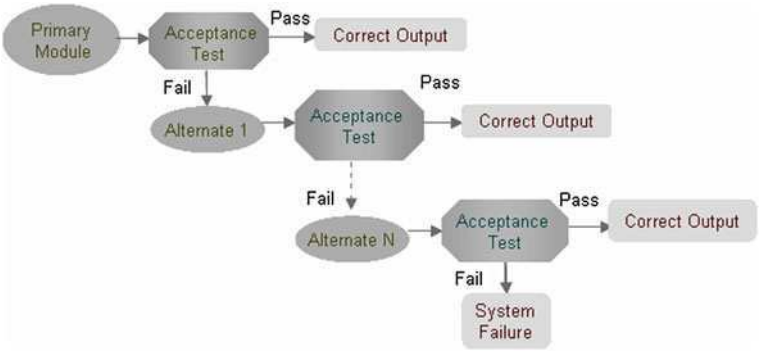


Fig. 12.3 Recovery block scheme

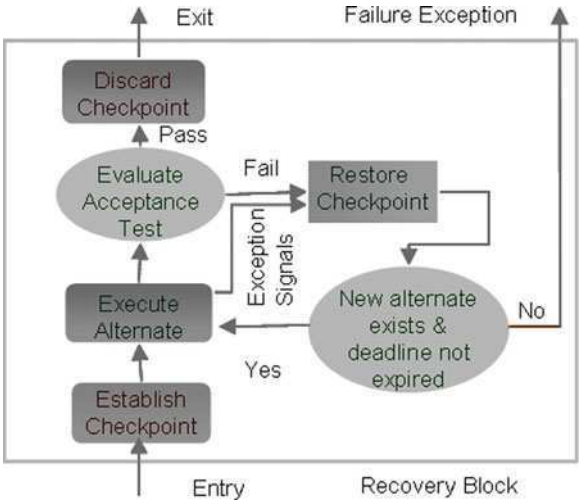


Fig. 12.4 Operation of recovery block

The significant difference in the recovery block approach from NVP is that only one version is executed at a time and the acceptability of results is decided by a test rather than by majority voting. While the advantage with the NVP is that average expected time of execution is lesser than the recovery block as all versions are executed simultaneously. For this reason often recovery block scheme is avoided for implementation in critical control software where real-time response is of great concern.

An important concern related to the implementation of fault tolerant schemes NVP or RB is that although it is sure that there is some degree of reliability improvement but it incurs a huge cost. One has to carry a tradeoff between the level of reliability desired and the cost of implementation before implementing any fault tolerant scheme.

12.2.3 Some Advanced Techniques

Many applications and varieties of both NVP and RB have been explored and developed by various researchers. Some of them also combine the features of both. Here we give a brief discussion of some of them.

12.2.3.1 Community Error Recovery

NVP has been researched thoroughly during the past years. The sources of failure of a NVP scheme are the common errors. Design diversity plays the major role in minimizing these types of faults. As already mentioned that NVP systems are used more successfully for implementation in the safety critical systems, they suffer from a drawback that voting at the end of the execution to decide the correct output may not be acceptable in such systems. As an alternate an alternative scheme called Community Error Recovery (CER) [11] is proposed. This scheme offers a higher degree of fault tolerance compared to the basic NVP scheme. In this scheme, comparisons of results are done at intermediate points; however, it requires synchronization of various versions at the comparison points.

12.2.3.2 Self-Checking Duplex Scheme

This scheme also adopts an intermediate voting. Generalization of this scheme is called N-program self-checking scheme [12]. Here each version is subject to an acceptance test or checking by comparison. When redundancy is implemented at the two levels, it is called self-checking duplex scheme. The scheme is built on the observation that if individual versions are made highly reliable, an ultra high reliability can be achieved merely by building only two versions simultaneously. Here whenever a particular version raises an exception the correct results are obtained from the remaining versions and the execution is continued, The approach is similar to the CER scheme with the difference that the online detection in the former is carried by an acceptance test rather than a comparison.

12.2.3.3 Distributed Execution of Recovery Blocks

Hecht et al. [13] described a distributed fault tolerant architecture, called the extended distributed recovery block (EDRB), for nuclear reactor control and safety functions. It relies on commercially available components and thus allows for continuous and inexpensive system enhancement. A useful feature of this approach is the relatively low runtime overhead it requires so that it is suitable for incorporation into real-time systems. The basic structure of the distributed recovery block is—the entire recovery block, two alternates with an acceptance test, fully replicated on the primary and backup hardware nodes. However, the roles of the

two alternate modules are not the same in the two nodes. The primary node uses the first alternate as the primary initially, whereas the backup node uses the second alternate as the initial primary. Outside of the EDRB, forward recovery can be achieved in effect; but the node affected by a fault must invoke backward recovery by executing an alternate for data consistency with the other nodes.

12.2.3.4 Consensus Recovery Blocks

The consensus recovery block (CRB) [14] is an attempt to combine the techniques used in the recovery block and NVP. It is claimed that the CRB technique reduces the importance of the acceptance test used in the recovery block and is able to handle the case where NVP would not be appropriate since there are multiple correct outputs. The CRB requires design and implementation of N variants of the algorithm, which are ranked (as in the recovery block) in the order of service and reliance. On invocation, all variants re-executed and their results submitted to an adjudicator, i.e. a voter (as used in NVP). The CRB compares pairs of results for compatibility. If two results are the same then the result is used as the output. If no pair can be found then the results of the variant with the highest ranking are submitted to an acceptance test. If this fails then the next variant is selected. This continues until all variants are exhausted or one passes the acceptance test. Scott et al. [15] developed reliability models for the RB, NVP and the CRB. In comparison, the CRB is shown to be superior to the other two. However, the CRB is largely based on the assumption that there are no common faults between the variants. In particular, if a matching pair is found, there is no indication that the result is submitted to the acceptance test, so a correlated failure in two variants could result in an erroneous output and would cause a catastrophic failure.

12.2.3.5 Retry Blocks with Data Diversity

A retry block developed by Ammann and Knight [2] is a modification of the recovery block scheme that uses data diversity instead of design diversity. Data diversity is a strategy that does not change the algorithm of the system (just retry), but does change the data that the algorithm processes. It is assumed that there are certain data, which will cause the algorithm to fail, and that if the data were re-expressed in a different, equivalent (or near equivalent) form the algorithm would function correctly. A retry block executes the single algorithm normally and evaluates the acceptance test. If the test passes, the retry block is complete. If the test fails, the algorithm executes again after the data have been re-expressed. The system repeats this process until it violates a deadline or produces a satisfactory output. The crucial elements in the retry scheme are the acceptance test and the data re-expression routine. Compared to design diversity, data diversity is relatively easy and inexpensive to implement. Although additional costs are incurred in the algorithm for data re-expression, data diversity requires only a single

implementation of a specification. Of course, the retry scheme is not generally applicable and its expression algorithm must be tailored to the individual problem at hand and it should be simple enough to eliminate the chance of design faults.

The techniques discussed above are not the only available fault tolerant techniques, but many more have been discussed by many researchers. A detailed discussion on fault tolerant schemes has been done in [16].

Many researchers in the field of software reliability engineering (SRE) have done excellent research to study the fault tolerant systems in many ways. Most of the research in this area is either on optimization problems of optimum selection of redundant components [17–23] or focus on software diversity modeling and dependability measures for specific types of software systems [11, 15, 24–27]. Some work has also been done to analyze reliability growth during testing and debugging for these systems. Study of reliability growth analysis has been done only for NVP systems [28–30]. In the next sections of this chapter we will discuss the software reliability growth models for the NVP systems in continuous and discrete time space and the problems of optimum selection of redundant components for recovery blocks, NVP systems and consensus RB.

12.3 Reliability Growth Analysis of NVP Systems

There are only a few studies carried out in the literature for the reliability growth analysis of NVP systems. An initial attempt has been made by Kanoun et al. [28] using a hyper-exponential model assuming a perfect debugging environment. The failure intensity function of the model is given as

$$h(t) = \frac{\omega\xi_{\text{sup}}\mathrm{e}^{-\xi_{\text{sup}}\mathrm{t}} + \omega\xi_{\text{inf}}\mathrm{e}^{-\xi_{\text{inf}}\mathrm{t}}}{\omega\mathrm{e}^{-\xi_{\text{sup}}t} + \bar{\omega}\mathrm{e}^{-\xi_{\text{inf}}t}}, \quad 0 \le \omega \le 1,\ \bar{\omega} \equiv 1 - \omega$$

where $\xi_{\text{sup}}, \xi_{\text{inf}}$ are hyper-exponential model parameters. $h(t)$ is nonincreasing with time for $0 \le \omega \le 1$ with $h(0) = \omega\xi_{\text{sup}}$ and $h(\infty) = \xi_{\text{inf}}$.

Building very high reliability is of extreme importance for fault tolerant software hence we must consider the effect of testing efficiency on the reliability growth modeling for these systems. An imperfect testing efficiency results in lowering the reliability growth of the system. Teng and Pham [31] proposed a NHPP-based software reliability growth model for a NVP system considering the effect of fault removal efficiency using Zhang et al. [32] testing efficiency model (see Sect. 3.4). Kapur et al. [30] proposed SRGM for NVP systems based on integrated generalized testing efficiency models in continuous (see Sect. 3.4) and discrete time space for application software that describes the reliability growth during testing under two types of imperfect debugging. The models were formulated for the 3VP systems, which are extendable to the NVP type with ease. Before we discuss the models in detail first we discuss the type of faults and the failure mechanisms of NVP systems.

12.3.1 Faults in NVP Systems

In the literature faults in a NVP system are classified into two categories [31]

- Common faults (CF) and
- Independent faults (IF)

Common-faults are located in at least two functionally equivalent modules of software versions. Although different versions may be developed independently using various design diversity techniques, it is expected that programmers are prone to making similar mistakes.

Independent-faults on the other hand are usually located in different or functionally distinct modules of different software versions.

CF are known to be more critical as compared to the IF. IF can be easily tolerated since, on a failure due to independent fault only the version containing that fault is expected to fail which is masked by the NVP system. But if the fault type is CF, which is expected to occur in multiple versions, then it is possible that on an input several versions might fail simultaneously. A failure by common-faults is called common-failure. The voting mechanism might choose the incorrect output resulting in the system failure. One more type of failure mode also exists in NVP systems. It is possible that an input results in failure of two or more software versions simultaneously due to independent faults. These failures by unrelated independent faults are called concurrent independent failures (CIF). However, the probability of their occurrence is very less. On the occurrence of these types of failures the voter is not able to make a correct decision resulting in the total system failure. The role of faults in NVP systems might change due to imperfect debugging, and some potential common faults might reduce to low-level common faults or independent faults. For example in a 3VP system if failure occurs in all the three versions due to CF and the removal process makes perfect fix only in two versions and imperfectly in one, it reduces to an IF. Figure 12.5 shows CF and IF in a 2VP system and Fig. 12.6 shows the various faults in a 3VP system.

Teng and Pham [31] further simplified the fault classification of common and independent faults as follows

- If at least two versions give identical but all wrong results, then the failures are caused by the common faults between versions.
- If at least two versions give dissimilar but wrong results, then the faults are caused by independent software faults.

The faults in a 3VP system (say version A, B, C) are CF in all three versions (of type ABC), CF in two versions (of type AB, AC, BC) and independent faults (of type A, B, C). The model formulated model equation for each type of fault separately since each fault type is independent of each other and that the mean value function of the failure and removal process of the 3VP system is defined as a whole.

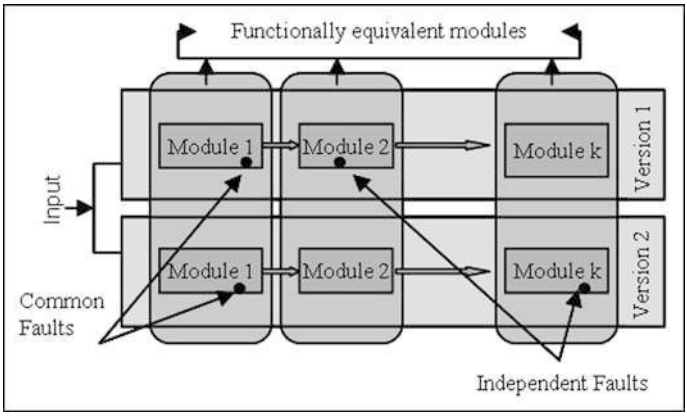
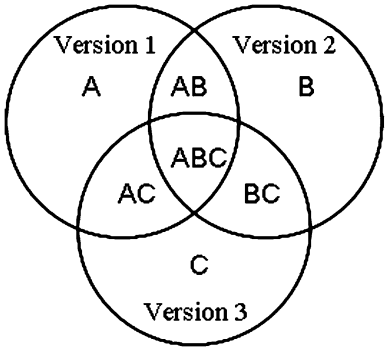


Fig. 12.5 Common and independent faults in a 2VP system

Fig. 12.6 Various faults in a 3VP system



12.3.2 Testing Efficiency Based Continuous Time SRGM for NVP System

Following notations and assumptions are defined for the model.

Notation

$m(t)$	Mean value function in the NHPP model, with $m(0) = 0$
a	Initial number of faults in the software at the time when testing of software starts
$a(t)$	Expected total fault content (remaining + removed) at time t
p	Probability of debugging of a fault perfectly
α	Constant rate of error generation
$m_r(t)$	Expected number of removals by time t
$m_f(t)$	Expected number of failures by time t
$b(t)$	Time dependent rate of fault removal per remaining faults

$R(x\|T)$	Pr{no failure occurs during $(T, T + x) \mid$ testing stops at T}
A, B, C	Independent faults in version 1, 2, 3, respectively
AB, BC, AC	Common faults between versions i and j, $i \neq j$, $i, j = 1, 2, 3$
ABC	Common faults in versions 1, 2, 3, respectively
$N_{\Phi,r}(t)$	Counting process denoting the number of faults of type $\Phi = ABC, \ldots, A$ removed up to time t
$N_{\Phi,f}(t)$	Counting process denoting the number of faults of type Φ detected up to time t
$N_c(t)$	$\sum N_Y(t)$, $Y = ABC, AB, AC, BC$, counting process for total CF detected up to time t.
$a_\Phi(t)$	Expected total fault content (remaining + removed) of fault of type Φ at time t
$a_\Phi(0)$	Initial number of type Φ faults in the system
b	Fault detection\removal rate per remaining fault at time t
α_i	Constant error generation rate during the debugging process in version $i = 1, 2, 3$
p_i	pr{of debugging of a fault perfectly in version $i = 1, 2, 3$}: $\bar{p}_i = 1 - p_i$; $p_{ij} = p_i.p_j$;$p_{ijk} = p_i \cdot p_j \cdot p_k$; $\bar{p}_{ij} = \bar{p}_i \cdot \bar{p}_j$; $\bar{p}_{ijk} = \bar{p}_i \cdot \bar{p}_j \cdot \bar{p}_k$
$X_\Phi(t)$	Number of faults of type Φ remaining in the system at time t
$R_{CF}(x\|T)$	Reliability of NVP system if only common faults are considered
$R_{IF}(x\|T)$	Reliability of NVP system if different versions contain only independent faults
$R_{NVP}(x\|T)$	Reliability of NVP system
λ_ψ	Failure intensity per pair of concurrent s-independent failures $\Psi = (A, B)\ (A, C), (B, C)$
$N_\psi(t)$	Counting process denoting the number of concurrent s-independent failures for $\Psi = (A, B), (A, C), (B, C)$ up to time t
$N_I(t)$	$\sum N_\Psi(t)$, $\Psi = (A,B), (A,C), (B,C)$ counting process for total concurrent s-independent failures up to time t
$m_{\eta,r}(t)$	$E[N_{\eta,r}(t)\}$, $\eta = A, \ldots, ABC, (A, B), (A, C), (B, C), c, I,c, I$
$m_{\eta,f}(t)$	$E[N_{\eta,f}(t)\}$, $\eta =$ A, …, ABC, $(A, B), (A, C), (B, C),c, I$
$h_\psi(t)$	$(d/dt)m_\psi(t)$;$\Psi = (A, B), (A, C), (B, C)$

Assumption

1. Failure observation/fault removal phenomenon is modeled by NHPP.
2. Faults remaining in the software cause software failures during execution.
3. Each time a failure is observed, an immediate effort takes place to isolate and remove the fault that has caused the failure.
4. Failure rate is equally affected by all the faults remaining in the software.
5. On a removal attempt a fault is removed perfectly with probability p, $0 \leq p \leq 1$.

6. During the fault removal process, new faults are generated with a constant probability α, $0 \leq \alpha \leq 1$.
7. Faster versions wait for the slower version to finish the execution prior to the voter's decision.
8. Software versions fail during execution caused by faults remaining in the software.
9. Two or more versions can fail on an input either due to the common faults or s-independent faults in different versions.
10. Some common faults may reduce to some low-level common faults or independent faults due to imperfect fault removal efficiency.
11. Probability of generating a common fault of type *ABC* while removing a fault of type *ABC* or of type *AB*, *AC*, *BC* while removing a fault type *AB*, *AC*, *BC*, respectively, is negligible and can be assumed to be zero.
12. The fault detection rate per remaining fault in each version is same for all kinds of faults and is a constant; $b(t) = b$.
13. Probability of a concurrent independent failure in all the versions i.e. *A*, *B*, *C* is negligible and can be assumed to be zero.
14. Intensity of concurrent s-independent failure for any two versions is proportional to the remaining number of s-independent pairs in those versions and each pair of remaining s-independent faults between versions has the same probability to be activated by some input.

12.3.2.1 Model Development

Using the generalized testing efficiency model [33] with a constant fault removal\detection rate b and a constant error generation rate α i.e.

$$b(t) = b \quad \text{and} \quad \frac{d}{dt}a(t) = \alpha\frac{d}{dt}m_r(t) \;\Rightarrow\; a(t) = a + \alpha m_r(t) \tag{12.3.1}$$

the mean value function of the removal phenomenon is given by

$$m_r(t) = \frac{a}{(1-\alpha)}(1 - \mathrm{e}^{-bp(1-\alpha)t}) \tag{12.3.2}$$

and the mean value function of the failure phenomenon using the relationship $m_r(t) = pm_f(t)$ is given as

$$m_f(t) = \frac{a}{p(1-\alpha)}\left(1 - \mathrm{e}^{-bp(1-\alpha)t}\right) \tag{12.3.3}$$

The mean value functions of the failure and removal phenomenon of different type of faults in a 3VP system are given as follows

Case 1 Common faults of type *ABC*

$$\frac{dm_{ABC,r}(t)}{dt} = p_{123}b\left(a_{ABC}(t) - m_{ABC,r}(t)\right) \tag{12.3.4}$$

where

$$a_{ABC}(t) = a_{ABC} \tag{12.3.5}$$

$$\frac{dm_{ABC,f}(t)}{dt} = b\big(a_{ABC}(t) - p_{123}m_{ABC,f}(t)\big) \tag{12.3.6}$$

Substituting (12.3.5) in (12.3.4) and solving under the initial $m_{ABC,r}(0) = 0$ we get

$$m_{ABC,r}(t) = a_{ABC}\big(1 - \mathrm{e}^{-bp_{123}t}\big) \tag{12.3.7}$$

and using $m_{ABC,r}(t) = p_{123}m_{ABC,f}(t)$ we have

$$m_{ABC,f}(t) = \frac{a_{ABC}}{p_{123}}\big(1 - \mathrm{e}^{-bp_{123}t}\big) \tag{12.3.8}$$

Case 2 Common faults of type AB, AC and BC

$$\frac{dm_{AB,r}(t)}{dt} = p_{12}b[a_{AB}(t) - m_{AB,r}(t)] \tag{12.3.9}$$

where

$$a_{AB}(t) = a_{AB} + \bar{p}_{12}p_3 m_{ABC,f}(t) \tag{12.3.10}$$

Substituting (12.3.10) in (12.3.9) and solving under the initial $m_{AB,r}(0) = 0$ we get

$$m_{AB,r}(t) = a_{AB}\big(1 - \mathrm{e}^{-bp_{12}t}\big) + \frac{\bar{p}_{12}a_{ABC}}{p_{12}\bar{p}_3}\big(p_3\mathrm{e}^{-bp_{12}t} - \mathrm{e}^{-bp_{123}t} + \bar{p}_3\big) \tag{12.3.11}$$

and using $m_{AB,r}(t) = p_{12}m_{AB,f}(t)$ we have

$$m_{AB,f}(t) = \frac{a_{AB}}{p_{12}}\big(1 - \mathrm{e}^{-bp_{12}t}\big) + \frac{\bar{p}_{12}a_{ABC}}{p_{12}^2\bar{p}_3}\big(p_3\mathrm{e}^{-bp_{12}t} - \mathrm{e}^{-bp_{123}t} + \bar{p}_3\big) \tag{12.3.12}$$

Mean value functions of removal (failure) phenomenon for faults of type AC and BC can be obtained similarly (see Appendix C).

Case 3 Independent faults of type A, B, and C

$$\frac{dm_{A,r}(t)}{dt} = p_1 b[a_A(t) - m_{A,r}(t)] \tag{12.3.13}$$

$$\begin{aligned} a_A(t) = a_A &+ \bar{p}_1\big(p_{23}m_{ABC,f}(t) + p_2 m_{AB,f}(t) + p_3 m_{AC,f}(t)\big) \\ &+ \alpha_1\left(\sum_{\upsilon} m_{\upsilon,r}(t)\right) \quad \upsilon = ABC,\ AB,\ AC,\ A \end{aligned} \tag{12.3.14}$$

Substituting (12.3.14) in (12.3.13) and solving under the initial $m_{A,r}(0) = 0$ we get

$$m_{A,r}(t) = \left(\frac{1}{1-\alpha_1}\right)\left\{\begin{array}{c} a_A + (1-p_1(1-\alpha_1)) \\ \left[\dfrac{a_{ABC}}{p_1} + \sum\limits_{i=2,V=B}^{3,C}\left(\dfrac{a_{AV}}{p_1} + \dfrac{\bar{p}_{1i}}{p_1^2 p_i} a_{ABC}\right)\right] \end{array}\right\}\left(1 - \mathrm{e}^{-b(1-\alpha_1)p_1 t}\right)$$
$$+ \left\{\frac{a_{ABC}(1-p_1(1-\alpha_1))}{(p_{23}-(1-\alpha_1))}\right\}\left\{\frac{1}{p_1} + \frac{\bar{p}_{12}}{p_1^2 p_2 \bar{p}_3} + \frac{\bar{p}_{13}}{p_1^2 p_3 \bar{p}_2}\right\}\left(\mathrm{e}^{-bp_{123}t} - \mathrm{e}^{-b(1-\alpha_1)p_1 t}\right)$$
$$+ \left(\frac{(1-p_1(1-\alpha_1))}{(p_2-(1-\alpha_1))}\right)\left\{\left(\frac{a_{AB}}{p_1} + \frac{\bar{p}_{12}p_3}{p_1^2 p_2 \bar{p}_3} a_{ABC}\right)\left(\mathrm{e}^{-bp_{12}t} - \mathrm{e}^{-b(1-\alpha_1)p_1 t}\right)\right\}$$
$$+ \left(\frac{(1-p_1(1-\alpha_1))}{(p_3-(1-\alpha_1))}\right)\left\{\left(\frac{a_{AC}}{p_1} + \frac{\bar{p}_{13}p_2}{p_1^2 p_3 \bar{p}_2} a_{ABC}\right)\left(\mathrm{e}^{-bp_{13}t} - \mathrm{e}^{-b(1-\alpha_1)p_1 t}\right)\right\} \tag{12.3.15}$$

and failure phenomenon is obtained using

$$m_{A,f}(t) = m_{A,r}(t)/p_1 \tag{12.3.16}$$

Similarly we can obtain the mean value functions of removal and failure phenomenon for faults of type B and C (see Appendix C).

12.3.2.2 System Reliability

Reliability of a NVP system can be calculated from a counting process that describes the total number of system failures by time t. A NVP system fails when a majority (more than half) of its versions fails assuming the voter mechanism to be 100% reliable. System failure is caused either on a common failure for e.g. when at least two versions of a 3VP system fail on a common failure or due to concurrent s-independent failure for e.g. when two or all the versions of a 3VP system fail on a concurrent s-independent failure [34]. However, the probability that all the three versions fail on the same input because of independent faults is negligible.

Counting Process of Common Failure Mode

For common failure mode (CFM) the counting process denoting the number of common failures up to time t is

$$N_c(t) \equiv N_{ABC,f}(t) + N_{AB,f}(t) + N_{AC,f}(t) + N_{BC,f}(t) \tag{12.3.17}$$

Equation (12.3.17) implies

$$E[N_c(t)] \equiv E\left[N_{ABC,f}(t) + N_{AB,f}(t) + N_{AC},f(t) + N_{BC,f}(t)\right]$$
$$\Rightarrow \quad m_{c,f}(t) \equiv m_{ABC,f}(t) + m_{AB,f}(t) + m_{AC,f}(t) + m_{BC,f}(t) \tag{12.3.18}$$

The probability that a 3VP system will not fail during $(T, T + x)$, given that the last failure occurred at T considering only common faults in the system, is

$$R_{CF}(x|T) = \mathrm{e}^{-\left(m_{c,f}(T+x)-m_{c,f}(T)\right)} \tag{12.3.19}$$

Counting Process of Concurrent Independent Failure Mode

From Assumptions 13 and 14

$$h_{(A,B)}(t) = \lambda_{(A,B)}\, X_A(t)\, X_B(t) \tag{12.3.20}$$

where

$$X_A(t) = a_A(t) - m_{A,,r}(t) \tag{12.3.21}$$

$$X_B(t) = a_B(t) - m_{B,,r}(t) \tag{12.3.22}$$

Hence the mean value function of number of CIF in versions A and B is

$$m_{(A,B),f}(t) = \int_0^t h_{(A,B)}(u)du \tag{12.3.23}$$

Similarly we obtain

$$m_{(A,C),f}(t) = \int_0^t h_{(A,C)}(u)du \quad \text{where } h_{(A,C)}(t) = \lambda_{(A,C)}\, X_A(t)\, X_C(t) \tag{12.3.24}$$

$$m_{(B,C),f}(t) = \int_0^t h_{(B,C)}(u)du \quad \text{where} \quad h_{(B,C)}(t) = \lambda_{(B,C)}\, X_B(t)\, X_C(t) \tag{12.3.25}$$

For CIF mode the counting process denoting the number of CIF up to time t is given as

$$N_I(t) \equiv N_{(A,B)}(t) + N_{(A,C)}(t) + N_{(B,C)}(t) \tag{12.3.26}$$

Equation (12.3.26) implies

$$m_{I,f}(t) \equiv m_{(A,B),f}(t) + m_{(A,C),f}(t) + m_{(B,C),f}(t) \tag{12.3.27}$$

Reliability of a 3VP system considering only independent faults in the system is

$$R_I(x|T) = \mathrm{e}^{-\left(m_{I,f}(T+x)-m_{I,f}(T)\right)} \tag{12.3.28}$$

Reliability of 3VP System

Since the common failure and concurrent independent failure modes (CIF) are independent, therefore total system reliability of a 3VP system using (12.3.19) and (12.3.28) is given by

$$R_{3VP}(x|T) = R_{CF}(x|T) \cdot R_I(x|T) = \mathrm{e}^{-\left(m_{c,f}(T+x)+m_{I,f}(T+x)-m_{c,f}(T)-m_{I,f}(T)\right)} \quad (12.3.29)$$

As already mentioned, only a few attempts have been made in the literature to analyze the reliability growth during testing and debugging of NVP systems. Most of these efforts are made in continuous time space. Recently Kapur et al. [35] analyzed the reliability growth of NVP systems in discrete time space. In this section we discuss the model in detail with the numerical application. Similar to the case of continuous SRGM, the SRGM in discrete time space is formulated for 3VP systems, based on an integrated testing efficiency model for independent software. The model can be extended to NVP systems equivalently.

12.3.3 A Testing Efficiency Based Discrete SRGM for a NVP System

The model integrates the effect of both imperfect debugging and error generation on reliability growth of software based on the following notations.

Notation

$m_r(n)$	Expected number of removals by the nth test input, with $m_r(0) = 0$
$m_f(n)$	Expected number of failures by the nth test input, with $m_f(0) = 0$
b	Constant rate of fault removal/detection per remaining faults
a	Initial number of faults in the software at the time when testing of software starts
$a(t)$	Expected total fault content (remaining + removed) at the nth test input, with $a(0) = a$
p	Probability of debugging of a fault perfectly
α	Constant rate of fault generation
$R(x\|N)$	pr{no failure occurs up to the execution of $(N + x)$th test case given that last failure occurred at the execution of Nth test case}
A, B, C	Independent faults in version 1, 2, 3, respectively
AB, BC, AC	Common faults (CF) between versions i and j, $i \neq j$, $i, j = 1, 2, 3$
ABC	Common faults in versions 1, 2, 3
$N_{\Phi,r}(n)$	Counting process denoting the number of faults of type Φ, $\Phi = ABC, \ldots, A$ removed by the nth test input
$N_{\Phi,f}(n)$	Counting process denoting the number of faults of type Φ, $\Phi = ABC, \ldots, A$ detected by the nth test input
$N_c(n)$	$\sum N_\Upsilon(n)$, $\Upsilon = ABC, AB, AC, BC$ counting process for total CF detected by the nth test input
$a(n), a_\Phi(n)$	Expected total fault content (remaining + removed) of fault of type Φ, $\Phi = ABC, \ldots, A$ at the nth test input
$a_\Phi(0)$	Initial number of type Φ, $\Phi = ABC, \ldots, A$ faults in the system
b	Fault detection\removal rate per remaining fault at the nth test input

α, α_i	Constant error generation rate during the debugging process in version i = 1, 2, 3, *respectively*
p, p_i	pr [of debugging of a fault perfectly in version i = 1, 2, 3] $\bar{p}_i = 1 - p_i$; $p_{ij} = p_i \cdot p_j$;$p_{ijk} = p_i \cdot p_j \cdot p_k$; $\bar{p}_{ij} = \bar{p}_i \cdot \bar{p}_j$; $\bar{p}_{ijk} = \bar{p}_i \cdot \bar{p}_j \cdot \bar{p}_k$
$X_\Phi(n)$	Number of faults of type Φ remaining in the system at the nth test input
$R_{CF}(x\|N)$	Reliability of NVP system if only common faults are considered
$R_{IF}(x\|N)$	Reliability of NVP system if different versions contain only independent faults
$R_{\mathrm{NVP}}(x\|N)$	Reliability of NVP system
λ_ψ	Failure intensity per pair of concurrent s-independent failures $\Psi = (A,B),(A,C),(B,C)$
$N_\psi(n)$	Counting process denoting the number of concurrent s-independent failures for $\Psi = (A,B),(A,C),(B,C)$ by the nth test input
$N_I(n)$	$\sum N_\Psi(t)$, $\Psi = (A,B),(A,C),(B,C)$ counting process for total concurrent independent failures by the nth test input
$m_{\eta,r}(n)$	$E[N_{\eta,r}(n)\}, \eta = A,\ldots,ABC,(A,B),(A,C),(B,C),c,I$
$m_{\eta,f}(n)$	$E[N_{\eta,f}(n)\}$, $\eta = A,\ldots,ABC,(A,B),(A,C),(B,C)\},c,I$
$h_\psi(n)$	Failure intensity function of concurrent s-independent failures $\Psi = (A,B),(A,C),(B,C)$

12.3.3.1 Model Development

Here a discrete testing efficiency SRGM with a constant fault removal\detection rate b and a constant error generation rate α is used to formulate the SRGM of 3VP systems.

The mean value function of the removal phenomenon is given as

$$m_r(n) = \frac{a}{1-\alpha}(1 - (1 - bp\delta(1-\alpha))^n) \tag{12.3.30}$$

and reliability of the software is

$$R(x|N) = \mathrm{e}^{-\left(m_f(N+x) - m_f(N)\right)} \tag{12.3.31}$$

Based on the Assumptions 1–14 in Sect. 12.3.2, the mean value functions of the failure and removal phenomenon for the different type of faults in a NVP system in discrete time space are given as follows

Case 1 Common faults of type *ABC*

$$\frac{\left(m_{ABC,r}(n+1) - m_{ABC,r}(n)\right)}{\delta} = p_{123} b\left(a_{ABC}(n) - m_{ABC,r}(n)\right) \tag{12.3.32}$$

where $a_{ABC}(n) = a_{ABC}$

$$\frac{\left(m_{ABC,f}(n+1) - m_{ABC,f}(n)\right)}{\delta} = b\left(a_{ABC}(n) - p_{123}m_{ABC,f}(n)\right) \tag{12.3.33}$$

Using the method of PGF under the initial condition $m_{ABC,r}(0) = 0$ we get

$$m_{ABC,r}(n) = a_{ABC}(1 - (1 - bp_{123}\delta)^n) \tag{12.3.34}$$

And using $m_{ABC,r}(n) = pm_{ABC,f}(n)$ we have

$$m_{ABC,f}(n) = \frac{a_{ABC}}{p_{123}}(1 - (1 - bp_{123}\delta)^n) \tag{12.3.35}$$

The equivalent continuous SRGM corresponding to (12.3.34) and (12.3.35) is obtained taking limit $\delta \to 0$ and defining $t = n\delta$ as given by Eqs. (12.3.7) and (12.3.8).

Case 2 Common faults of type AB, AC and BC

$$\frac{\left(m_{AB,r}(n+1) - m_{AB,r}(n)\right)}{\delta} = p_{12}b\left(a_{AB}(n) - m_{AB,r}(n)\right) \tag{12.3.36}$$

where

$$a_{AB}(n) = a_{AB} + \bar{p}_{12}p_3 m_{ABC,f}(n) \tag{12.3.37}$$

Substituting (12.3.37) in (12.3.36) and solving using the method of PGF under the initial condition $m_{AB,r}(0) = 0$ we get

$$\begin{aligned} m_{AB,r}(n) = {} & a_{AB}(1 - (1 - bp_{12}\delta)^n) \\ & + \frac{a_{ABC}\bar{p}_{12}}{p_{12}\bar{p}_3}(p_3(1 - bp_{12}\delta)^n - (1 - bp_{123}\delta)^n + \bar{p}_3) \end{aligned} \tag{12.3.38}$$

and using $m_{AB,r}(n) = pm_{AB,f}(n)$we have

$$\begin{aligned} m_{AB,f}(t) = {} & \frac{a_{AB}}{p_{12}}(1 - (1 - bp_{12}\delta)^n) \\ & + \frac{a_{ABC}\bar{p}_{12}}{p_{12}^2\bar{p}_3}(p_3(1 - bp_{12}\delta)^n - (1 - bp_{123}\delta)^n + \bar{p}_3) \end{aligned} \tag{12.3.39}$$

Mean value functions of failure and removal phenomenon for faults of type AC and BC can be obtained similarly (see Appendix C). The equivalent continuous

SRGM corresponding to Eqs. (12.3.38) and (12.3.39) is obtained taking limit $\delta \to 0$ and defining $t = n\delta$, as given by Eqs. (12.3.11) and (12.3.12).

Case 3 Independent faults of type A, B and C

$$\frac{\big(m_{A,r}(n+1) - m_{A,r}(n)\big)}{\delta} = p_1 b\big(a_A(n) - m_{A,r}(n)\big) \tag{12.3.40}$$

$$\begin{aligned} a_A(n) = a_A &+ \bar{p}_1\big(p_{23} m_{ABC,f}(n) + p_2 m_{AB,f}(n) + p_3 m_{AC,f}(n)\big) \\ &+ \alpha_1\big(m_{ABC,r}(n) + m_{AB,r}(n) + m_{AC,r}(n) + m_{A,r}(n)\big) \end{aligned} \tag{12.3.41}$$

Substituting (12.3.41) in (12.3.40) and solving under the initial $m_{A,r}(0) = 0$ using the method of PGF we get

$$\begin{aligned} m_{A,r}(n) = \left(\frac{1}{1-\alpha_1}\right) &\left\{ a_A + (1 - p_1(1-\alpha_1) \right. \\ &\left. \left[\frac{a_{ABC}}{p_1} + \sum_{(i,V)=(2,B),(3,C)} \left(\frac{a_{AV}}{p_1} + \frac{\bar{p}_{1i}}{p_1^2 p_i} a_{ABC} \right) \right] \right\} \\ &\times (1 - (1 - b p_1 \delta(1-\alpha_1))^n) \\ &+ \left\{ \frac{a_{ABC}(1 - p_1(1-\alpha_1))}{(p_{23} - (1-\alpha_1))} \right\} \left\{ \frac{1}{p_1} + \frac{\bar{p}_{12}}{p_1^2 p_2 \bar{p}_3} + \frac{\bar{p}_{13}}{p_1^2 p_3 \bar{p}_2} \right\} \\ &\times ((1 - b p_{123}\delta)^n - (1 - b p_1 \delta(1-\alpha_1))^n) \\ &+ \left(\frac{(1 - p_1(1-\alpha_1))}{(p_2 - (1-\alpha_1))} \right) \left\{ \left(\frac{a_{AB}}{p_1} + \frac{\bar{p}_{12} p_3}{p_1^2 p_2 \bar{p}_3} a_{ABC} \right) \right. \\ &\left. \times ((1 - b p_{12}\delta)^n - (1 - b p_1 \delta(1-\alpha_1))^n) \right\} \\ &+ \left(\frac{(1 - p_1(1-\alpha_1))}{(p_3 - (1-\alpha_1))} \right) \left\{ \left(\frac{a_{AC}}{p_1} + \frac{\bar{p}_{13} p_2}{p_1^2 p_3 \bar{p}_2} a_{ABC} \right) \right. \\ &\left. \times ((1 - b p_{13}\delta)^n - (1 - b p_1 \delta(1-\alpha_1))^n) \right\} \end{aligned} \tag{12.3.42}$$

and using

$$m_{A,r}(n) = p m_{A,f}(n) \tag{12.3.43}$$

we can obtain the mean value function for the failure phenomenon of fault type A. Similarly we can obtain the mean value functions of removal and failure

phenomenon for faults of type B and C (see Appendix C). The equivalent continuous SRGM for the common faults of type A, B, C can also be obtained taking limit $\delta \to 0$ and defining $t = n\delta$.

12.3.3.2 System Reliability

A NVP system fails when a majority (more than half) of its versions fails assuming the voter mechanism to be 100% reliable. System failure is caused either on a common failure or due to concurrent s-independent failure. However, the probability that all the three versions fail on the same input because of independent faults is negligible.

Counting Process of Common Failure Mode

For CFM the counting process denoting the number of common failures up to the execution of nth test input is

$$N_c(n) \equiv N_{ABC,f}(n) + N_{AB,f}(n) + N_{AC,f}(n) + N_{BC,f}(n) \tag{12.3.44}$$

Equation (12.3.44) implies

$$\begin{aligned} E[N_c(n)] &\equiv E\left[N_{ABC,f}(n) + N_{AB,f}(n) + N_{AC,f}(n) + N_{BC,f}(n)\right] \\ \Rightarrow m_{c,f}(n) &\equiv m_{ABC,f}(n) + m_{AB,f}(n) + m_{AC,f}(n) + m_{BC,f}(n) \end{aligned} \tag{12.3.45}$$

The probability that a 3VP system will not fail between the execution of Nth and $(N + x)$th test cases, given that the latest failure occurred at the execution of Nth test case considering only common faults in the system, is

$$R_{CF}(x|N) = \mathrm{e}^{-\left(m_{c,f}(N+x) - m_{c,f}(N)\right)} \tag{12.3.46}$$

Counting Process of CIF Mode

Similar to the case of continuous time SRGM

$$h_{(A,B)}(n) = \lambda_{(A,B)} X_A(n)\, X_B(n) \tag{12.3.47}$$

where

$$X_A(n) = a_A(n) - m_{A,r}(n) \tag{12.3.48}$$

$$X_B(n) = a_B(n) - m_{B,r}(n) \tag{12.3.49}$$

Hence the mean value function of number of CIF in versions A and B is

$$m_{(A,B),f}(n) = \sum_{i=0}^{n} h_{(A,B)}(i) \tag{12.3.50}$$

Similarly we obtain

$$m_{(A,C),f}(n) = \sum_{i=0}^{n} h_{(A,C)}(i) \quad \text{where } h_{(A,C)}(n) = \lambda_{(A,C)}\, X_A(n)\, X_C(n) \tag{12.3.51}$$

$$m_{(B,C),f}(n) = \sum_{i=0}^{n} h_{(A,B)}(i) \quad \text{where } h_{(B,C)}(n) = \lambda_{(B,C)}\, X_B(n)\, X_C(n) \tag{12.3.52}$$

For CIF mode the counting process denoting the number of common failures up to the execution of nth test input is

$$N_I(n) \equiv N_{(A,B)}(n) + N_{(A,C)}(n) + N_{(B,C)}(n) \tag{12.3.53}$$

Equation (12.3.53) implies

$$m_{I,f}(n) \equiv m_{(A,B),f}(n) + m_{(A,C),f}(n) + m_{(B,C),f}(n) \tag{12.3.54}$$

Reliability of a 3VP system considering only s-independent faults in the system is

$$R_I(x|N) = \mathrm{e}^{-\left(m_{I,f}(N+x) - m_{I,f}(N)\right)} \tag{12.3.55}$$

Reliability of 3VP System

Since the common failure and concurrent s-independent failure modes are s-independent, therefore total system reliability of a 3VP system using (12.3.46) and (12.3.55) is given by

$$R_{3VP}(x|N) = R_{CF}(x|N) \cdot R_I(x|N) = \mathrm{e}^{-\left(m_{c,f}(N+x) + m_{I,f}(N+x) - m_{c,f}(N) - m_{I,f}(N)\right)} \tag{12.3.56}$$

12.3.4 Parameter Estimation and Model Validation

The successful application of a software reliability growth model depends heavily on the quality of failure data collected. The parameters of the SRGM are estimated based upon these data. Mostly the method of maximum likelihood estimation or least square is used to estimate the unknown parameters of the nonlinear models. The system failure in a 3VP system can be due to common fault in all the three versions or due to fault in any two versions. Hence the regression module of SPSS is not sufficient to estimate the unknown parameters of the models discussed in this chapter. Conditional nonlinear regression (CNLR) method in SPSS solves this problem and it can conditionally determine the failure process of the 3VP system. This method is used in our analysis to find the unknown parameters of the models.

Table 12.2 Simulated failure data for a 3VP system

Fault	Failure time			Fault	Failure time		
#	*A*	*B*	*C*	#	*A*	*B*	*C*
1	1	2.4	1.3	15	35.2	25	25.2
2	3.5	3.5	2.4	16	39	26.9	29.5
3	6.3	4	4.9	17	42	29.5	30
4	7	4.7	6.3	18	45.8	35.2	32.8
5	9.1	6.3	7	19	47.1	39.5	34
6	10	7.2	8.6	20	48	43.3	35.2
7	14	8.6	10.3	21	50	45.8	38.4
8	18	10	12.8	22	51	48.1	47.1
9	20.3	12.5	15.1	23	52.5	52.6	50.6
10	21.6	14.6	17.6	24	54.3	58.7	52.5
11	23	15	20.4	25	54.8	59.2	55.3
12	27	18	21.6	26	57.6	60	59.2
13	29	19.7	21.9	27	59.2	63.3	60.2
14	30.7	20.2	23.5	28	60.6	67	64

12.3.4.1 Data Analysis for the Continuous Time Model

Failure Data Sets

Data analysis for this model is shown for two data sets, one for the 2VP system and other for the 3VP system. The observed failure data of a 2VP system are for a fault tolerant software control logic of a water reservoir control (WRC) system. 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 by using VP software control logic with WRC system [31]. The testing data for the WRC system are available for 100 test periods and during this period 26 failures were observed. Some of them were of common types while the others were of independent type.

Failure data set for a real 3VP system was unavailable, so Kapur et al. [30] simulated a failure data set for the 3VP system. The simulated data are tabulated in Table 12.2.

Data Analysis of a 2VP System

It is reasonable to assume sufficiently high testing efficiency. Therefore in order to simplify the estimation process $p_1 = p_2$ are assumed to be known, equal to 0.9. Generally, for a NVP system $N \geq 3$, so that a voting mechanism can be applied to

choose the correct output. However, for a 2VP system the reliability of the voter is assumed to be one and is self decision maker, i.e., the voter can decide which version(s) has failed on a failure. Further a 2VP system fails only when both of its versions fail on a same input. The observed failure data for WRC system show some data points at which both the versions fail simultaneously. At these time points system failure is observed which may be either due to common faults or independent common faults. Here it is assumed that each system failure is only due to common faults i.e. no system failure occurs due to concurrent independent failure. A 2VP system contains common faults of type *AB* and independent faults of type *A* and *B*. The mean value functions of the failure phenomenon of a 2VP system are derived similar to 3VP system and are given by the following equations.

$$m_{AB,f}(t) = \frac{a_{AB}}{p_{12}}\left(1 - \mathrm{e}^{-bp_{12}t}\right) \tag{12.3.57}$$

$$m_{A,f}(t) = \frac{1}{p_1}\left\{\begin{array}{l} \dfrac{a_A}{1-\alpha_1}\left(1 - \mathrm{e}^{-b(1-\alpha_1)p_1 t}\right) \\ + \dfrac{a_{AB}(1 - p_1(1-\alpha_1))}{p_1(1-\alpha_1)}\left(1 + \left\{\dfrac{(1-\alpha_1)e^{-bp_{12}t} - p_2\mathrm{e}^{-\mathrm{b}(1-\alpha_1)\mathrm{p_1 t}}}{p_2 - (1-\alpha_1)}\right\}\right) \end{array}\right\} \tag{12.3.58}$$

Similarly the mean value function of failure phenomenon for faults of type *B* can be obtained. Total system reliability of a 2VP system is given as

$$R_{2VP}(x|T) = R_{CF}(x|T) \cdot R_I(x|T) = \mathrm{e}^{-\left(m_{AB,f}(T+x)+m_{(A,B),f}(T+x)-m_{AB,f}(T)-m_{(A,B),f}(T)\right)} \tag{12.3.59}$$

Since failure in both versions on an input is assumed only due to common faults therefore the system reliability is

$$R_{2VP}(x|T) = R_{CF}(x|T) = \mathrm{e}^{-\left(m_{AB,f}(T+x)-m_{AB,f}(T)\right)} \tag{12.3.60}$$

Table 12.3 Estimated values of unknown parameters and their standard errors

Proposed model	a_{AB}	a_A	a_B	b	α_1	α_2
Estimated values	8.59	16	19.83	0.0194	0.04	1.00E-04
Standard error	4.09	12.23	15.31	0.0149	0.547	0.566

Table 12.4 Variance–covariance matrix

	a_{AB}	a_A	a_B	b	α_1	α_2
a_{AB}	16.78	47.47	60.18	−0.06	−1.93	−2.16
a_A	47.47	149.57	182.02	−0.18	−6.39	−6.51
a_B	60.18	182.02	234.4	−0.23	−7.37	−8.53
b	−0.06	−0.18	−0.23	0	0.01	0.01
α_1	−1.93	−6.39	−7.37	0.01	0.3	0.26
α_2	−2.16	−6.51	−8.53	0.01	0.26	0.32

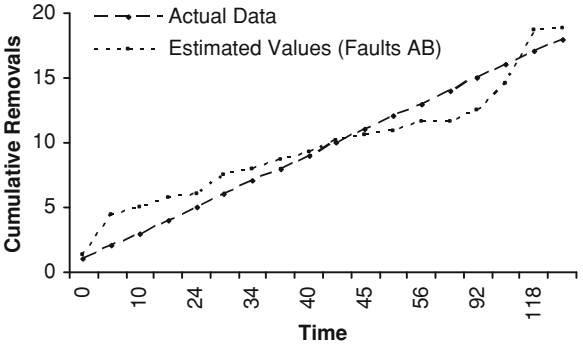


Fig. 12.7 Goodness of fit curve for fault type *AB*

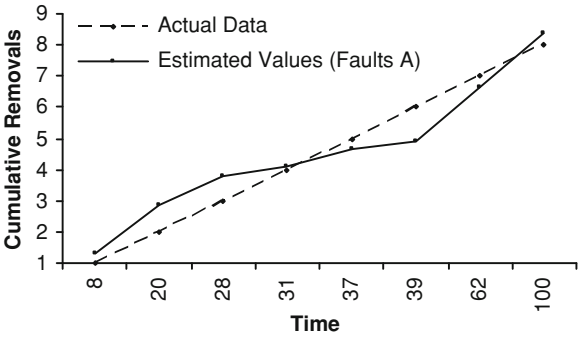


Fig. 12.8 Goodness of fit curve for fault type *A*

Estimating the parameters of the 2VP system using CNLR function of SPSS software package, the estimates of the unknown parameters and their respective standard error as shown in Table 12.3 are obtained. The variance–covariance matrix of the parameters for the proposed model is given in Table 12.4. The goodness of fit curves for the fault type *AB*, *A* and *B* are illustrated graphically in Figs. 12.7, 12.8, and 12.9, respectively. The system reliability curve is shown in Fig. 12.10.

Data Analysis of a 3VP System

From the simulated failure data of 3VP system (Table 12.2) failure times and number of faults for each type of faults (*ABC, …, C*) can be extracted. We assume that each system failure is only due to common faults (fault type *ABC, AB, AC, BC*) i.e. no system failure due to concurrent independent failures. In the data analysis of a 2VP we have assumed $p_1 = p_2 = 0.9$. In practice we may expect these parameters to be different for each version since different and independent teams may debug the software. Therefore from this point of view in the data analysis of a 3VP system $p_1 = 0.9$, $p_2 = 0.92$, $p_3 = 0.89$ are taken. Since the expressions of the mean value functions are very large, therefore for the sake of

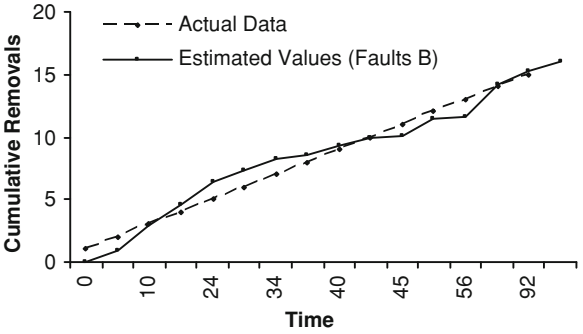


Fig. 12.9 Goodness of fit curve for fault type *B*

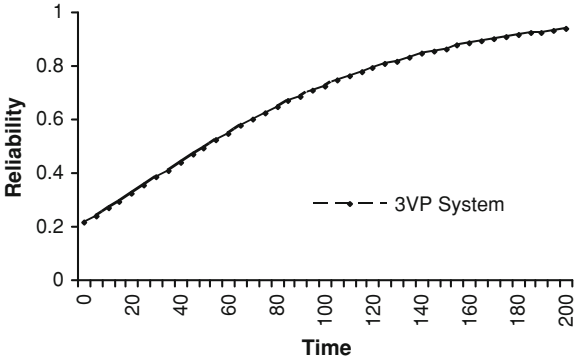


Fig. 12.10 System reliability growth

Table 12.5 Estimated values of unknown parameters and their standard errors

Proposed model	a_{ABC}	a_{AB}	a_{AC}	a_{BC}	a_A	a_B	a_C	b
Estimated values	41.7	47.8	59.26	54.29	140	128	116	0.00188
Standard error	18.91	20.72	25.21	23.59	60.51	53.82	49.39	0.0009

Table 12.6 Variance–covariance matrix

	a_{ABC}	a_{AB}	a_{AC}	a_{BC}	a_A	a_B	a_C	b
a_{ABC}	357.59	362.59	446.69	414.41	1087.49	966.85	887.64	−0.0162
a_{AB}	362.59	429.32	501.14	464.88	1218.91	1083.93	995.01	−0.0182
a_{AC}	446.69	501.14	635.54	572.7	1501.81	1336.04	1223.71	−0.0224
a_{BC}	414.41	464.88	572.7	556.49	1393.46	1238.89	1137.38	−0.0207
a_A	1087.49	1218.91	1501.81	1393.46	3661.46	3250.79	2982.31	−0.0544
a_B	966.85	1083.93	1336.04	1238.89	3250.79	2896.59	2652.85	−0.0484
a_C	887.64	995.01	1223.71	1137.38	2982.31	2652.85	2439.37	−0.0444
b	−0.0162	−0.0182	−0.0224	−0.0207	−0.0544	−0.0484	−0.0444	8.10E-07

simplicity the fault generation parameters are also assumed to be known. It is reasonable to assume sufficiently high testing efficiency, hence a small value of the fault generation rate is observed. Assume $\alpha_1 = 0.055$, $\alpha_2 = 0.04$ and $\alpha_3 = 0.05$.

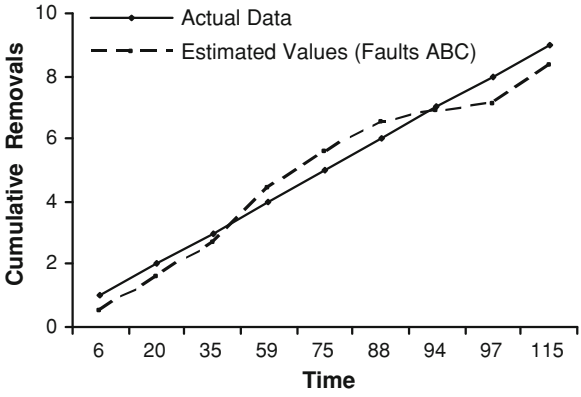


Fig. 12.11 Goodness of fit curve for fault type *ABC*

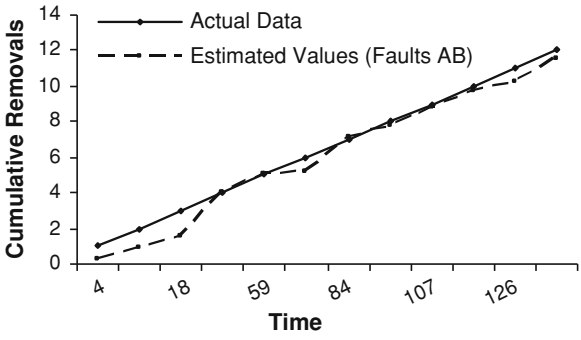


Fig. 12.12 Goodness of fit curve for fault type *AB*

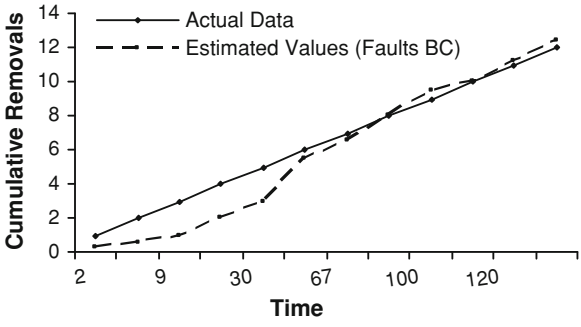


Fig. 12.13 Goodness of fit curve for fault type *AC*

These values are taken for the sake of illustration and can be determined form the past failure data and experience. The other unknown parameters are estimated using CNLR estimation method and estimation results and standard error of the estimated parameters are tabulated in Table 12.5. The variance–covariance matrix is given in Table 12.6. Goodness of fit curve for fault type *ABC*, *AB*, *AC*, *BC*, *A*, *B* and *C* are given in Figs. 12.11, 12.12, 12.13, 12.14, 12.15, 12.16, and 12.17 and system reliability curve is shown in Fig. 12.18.

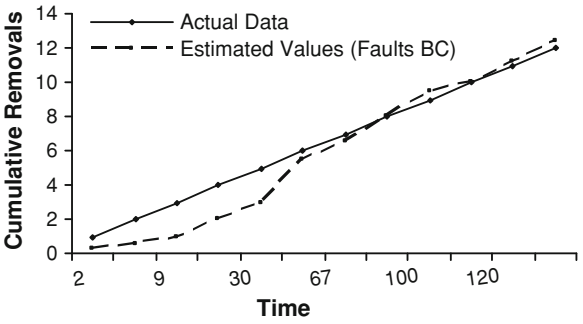


Fig. 12.14 Goodness of fit curve for fault type *BC*

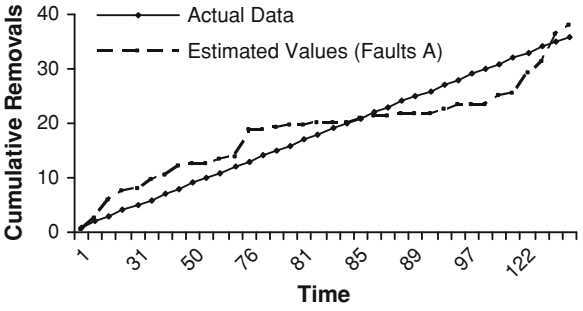


Fig. 12.15 Goodness of fit curve for fault type *A*

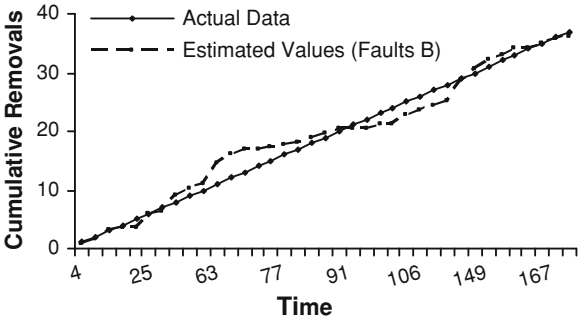


Fig. 12.16 Goodness of fit curve for fault type *B*

From the estimation results and goodness of fit curves of 2VP and 3VP systems it can be seen that the continuous time model fairly estimates the failure phenomenon of the NVP systems. In the beginning of this section we discussed the importance of reliability improvement and building fault tolerance in real-time safety and critical systems. The importance of building highest possible reliability in these systems also necessitates an accurate estimation of the achieved reliability. The continuous time model describes failure occurrence and fault removal in continuous time space and adopts time (the CPU time or calendar time) as a unit of fault detection period. As we know discrete time models (refer to Chap. 9), which adopt the number of test occasions/cases as a unit of fault detection period

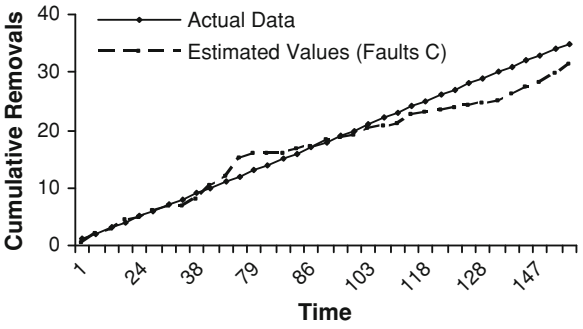


Fig. 12.17 Goodness of fit curve for fault type *C*

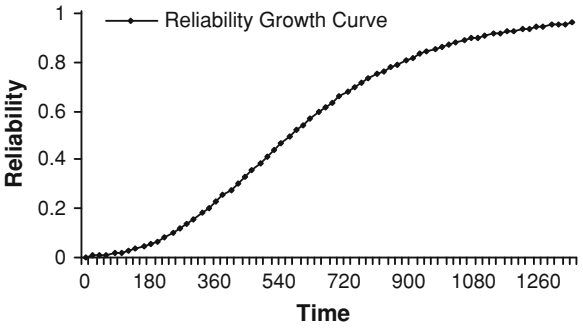


Fig. 12.18 System reliability growth

sometimes, provide better estimates of testing process when observed data sets are defined in discrete time space. So we also validate the discrete time model discussed in the previous section in the following section.

12.3.4.2 Parameter Estimation and Model Validation of Discrete SRGM

Similar to the case of continuous SRGM conditional nonlinear regression (CNLR) method is used to estimate the unknown parameters of a 2 and 3VP system on simulated failure data sets tabulated in Tables 12.7 and 12.8, respectively. The real life failure data were not available.

Data Analysis of a 2VP System

The model equations are very lengthy therefore in order to simplify the estimation process p_1 and p_2 are assumed to be known values. Since it is reasonable to assume sufficiently high testing efficiency therefore $p_1 = 0.9$ and $p_2 = 0.94$ are taken. A 2VP system fails only when both of its versions fail on a same input. The failure data for 2VP system show some data points at which both the versions fail simultaneously. At these time points system failure is observed which may be either due to common faults or independent common faults. Here also it is assumed that

Table 12.7 Simulated discrete failure data (2VP system)

Fault #	Test case #		Fault #	Test case #		Fault #	Test case #		Fault #	Test case #		Fault #	Test case #	
	A	*B*		*A*	*B*		*A*	*B*		*A*	*B*		*A*	*B*
1	2	4	15	20	18	29	34	29	43	44	38	57	80	60
2	3	8	16	24	20	30	36	31	44	45	39	58	92	62
3	8	13	17	28	23	31	37	35	45	54	41	59	99	98
4	10	14	18	29	25	32	39	36	46	56	42	60		99
5	16	17	19	31	27	33	40	37	47	62	46	61		100

Table 12.8 Simulated failure data (discrete 3VP SRGM)

Fault #	Failure time			Fault	Failure time			Fault	Failure time			Fault	Failure time		
	A	B	C	#	A	B	C	#	A	B	C	#	A	B	C
1	1	3	3	14	40	41	47	27	86	84	80	40	132	137	139
2	3	6	5	15	43	45	49	28	88	95	84	41	135	140	143
3	6	7	8	16	47	47	54	29	89	93	89	42	139	146	148
4	8	8	9	17	49	50	56	30	95	99	93	43	146	149	151
5	11	9	15	18	55	53	59	31	99	101	99	44	148	153	157
6	15	11	18	19	58	55	60	32	103	104	103	45	153	157	162
7	20	13	20	20	60	62	62	33	107	105	108	46	158	161	165
8	23	19	24	21	61	63	63	34	110	107	115	47	159	170	168
9	24	20	28	22	63	69	67	35	112	115	117	48	162	178	171
10	27	26	29	23	71	74	70	36	117	120	124	49	164		174
11	28	29	33	24	74	76	72	37	120	122	132	50	169		
12	32	32	39	25	79	80	77	38	126	127	136				
13	34	34	45	26	84	83	79	39	129	132	137				

each system failure is only due to common faults, i.e., no system failure occurs due to concurrent independent failure. The mean value functions of the failure phenomenon of a 2VP system can be derived similar to 3VP system. The mean value functions of the failure phenomenon due to fault type *AB*, *A*, and *B* are as follows

$$m_{AB,f}(n) = \frac{a_{AB}}{p_{12}}(1 - (1 - bp_{12}\delta)^n) \tag{12.3.61}$$

$$m_{A,f}(t) = \frac{1}{p_1}\left\{ \begin{array}{l} \dfrac{a_A}{1-\alpha_1}(1-(1-bp_1(1-\alpha_1)\delta)^n) \\ +\dfrac{a_{AB}(1-p_1(1-\alpha_1))}{p_1(1-\alpha_1)}\left(1+\left\{\dfrac{(1-\alpha_1)(1-bp_{12}\delta)^n - p_2(1-bp_1(1-\alpha_1)\delta)^n}{p_2-(1-\alpha_1)}\right\}\right) \end{array} \right\}$$

Similarly we can obtain the mean value function of failure phenomenon for faults of type *B* and total system reliability of a 2VP system is given as

$$R_{2VP}(x|N) = R_{CF}(x|N) \cdot R_I(x|N) = e^{-\left(m_{AB,f}(N+x)+m_{(A,B),f}(N+x)-m_{AB,f}(N)-m_{(A,B),f}(N)\right)} \quad (12.3.63)$$

Since it is assumed that failure in both versions on an input is only due to common faults therefore the system reliability would be given by

$$R_{2VP}(x|N) = R_{CF}(x|N) = e^{-\left(m_{AB,f}(N+x)-m_{AB,f}(N)\right)} \quad (12.3.64)$$

Estimating the parameters of the 2VP system using CNLR function of SPSS software package we obtain the estimates of the unknown parameters and their respective standard errors as shown in Table 12.9. The variance–covariance matrix is given in Table 12.10. The goodness of fit curve for the fault type *AB*, *A*, and *B* are illustrated graphically in Figs. 12.19, 12.20 and 12.21, respectively. The system reliability curve is shown in Fig. 12.22.

Table 12.9 Estimated values of unknown parameters and their standard errors (discrete 2VP SRGM)

Proposed model	a_{AB}	a_A	a_B	b	α_1	α_2
Estimated values	9.527	15.126	17.458	0.019536	0.0043	0.01023
Standard error	2.2275	5.728829	6.51	0.007148	0.24767	0.24175

Table 12.10 Variance–covariance matrix (discrete 2VP SRGM)

	a_{AB}	a_A	a_B	b	α_1	α_2
a_{AB}	4.96	12.05	13.90	−0.0155	−0.4736	−0.4994
a_A	12.05	32.82	35.84	−0.0398	−1.3612	−1.2861
a_B	13.90	35.84	42.38	−0.0460	−1.3990	−1.5522
B	−0.02	−0.04	−0.05	0.0001	0.0016	0.0016
α_1	−0.47	−1.36	−1.40	0.0016	0.0613	0.0502
α_2	−0.50	−1.29	−1.55	0.0016	0.0502	0.0584

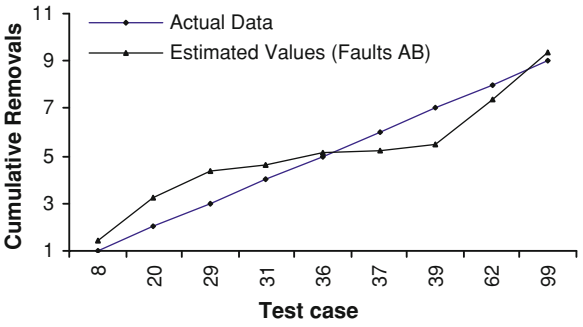


Fig. 12.19 Goodness of fit curve for fault type *AB* (discrete 2VP SRGM)

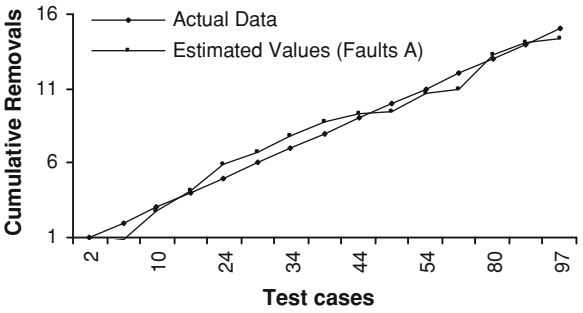


Fig. 12.20 Goodness of fit curve for fault type *A* (discrete 2VP SRGM)

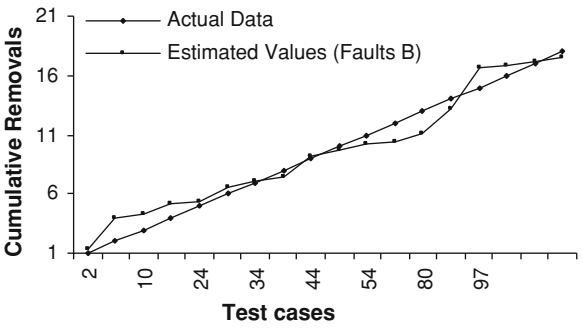


Fig. 12.21 Goodness of fit curve for fault type *B* (discrete 2VP SRGM)

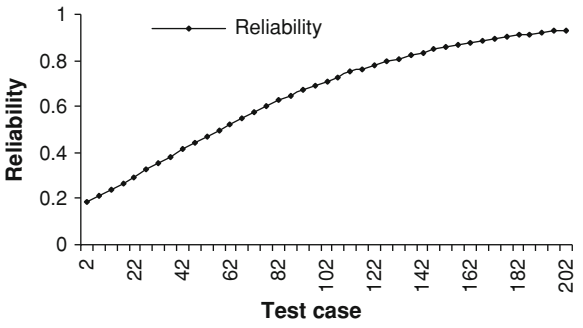


Fig. 12.22 System reliability growth (discrete 2VP SRGM)

Data Analysis of a 3VP System

From the data in Table 12.8 failure times and number of faults for each type of fault (*ABC, …, C*) can be extracted. Here also it is assumed that each system failure is only due to common faults (fault type *ABC*, *AB*, *AC*, *BC*). In the data analysis of a 3VP system $p_1 = 0.9$, $p_2 = 0.92$, $p_3 = 0.89$ are taken. Since the expressions of the mean value functions is very large, therefore for the sake of simplicity the fault generation parameters are also assumed to be known. It is reasonable to assume sufficiently high testing efficiency hence a small value of the fault generation rate is observed. $\alpha_1 = 0.055$, $\alpha_2 = 0.04$ and $\alpha_3 = 0.05$ are assumed. These values are taken for the sake of illustration and can be determined

Table 12.11 Estimated values of unknown parameters and their standard errors (discrete 3VP SRGM)

Proposed model	a_{ABC}	a_{AB}	a_{AC}	a_{BC}	a_A	a_B	a_C	b
Estimated values	28.07	36.47	37.396	34.39	59.54	58.09	55.06	0.0023
Standard error	7.083	8.89	9.02	8.34	14.63	14.02	13.54	0.0006

Table 12.12 Variance–covariance matrix (discrete 3VP SRGM)

	a_{ABC}	a_{AB}	a_{AC}	a_{BC}	a_A	a_B	a_C	b
a_{ABC}	50.17	60.61	61.75	56.85	100.84	96.48	93.30	−0.0042
a_{AB}	60.61	79.03	78.84	72.57	128.49	123.12	118.93	−0.0054
a_{AC}	61.75	78.84	81.36	73.96	130.95	125.50	121.17	−0.0055
a_{BC}	56.85	72.57	73.96	69.56	120.56	115.49	111.52	−0.0050
a_A	100.84	128.49	130.95	120.56	214.04	204.60	197.58	−0.0089
a_B	96.48	123.12	125.50	115.49	204.60	196.56	189.32	−0.0085
a_C	93.30	118.93	121.17	111.52	197.58	189.32	183.33	−0.0082
b	−0.0042	−0.0054	−0.0055	−0.0050	−0.0089	−0.0085	−0.0082	3.7E-07

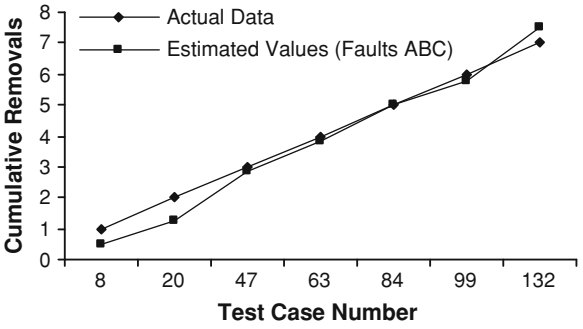


Fig. 12.23 Goodness of fit curve for fault type *ABC* (discrete 3VP SRGM)

from the past failure data and experience. The other unknown parameters are estimated using CNLR estimation method and estimation results and standard error of the estimated parameters are tabulated in Table 12.11. The variance-covariance matrix is given in Table 12.12. Goodness of fit curves for fault type *ABC*, *AB*, *AC*, *BC*, *A*, *B*, and *C* are given in Figs. 12.23, 12.24, 12.25, 12.26, 12.27, 12.28, and 12.29 and system reliability curves is shown in Fig. 12.30.

From the estimation results and goodness of fit curves of 2 and 3VP systems it can be seen that the model can fairly predict the failure phenomenon of the NVP systems.

12.4 COTS Based Reliability Allocation Problem

All fault tolerance techniques provide some degree of reliability improvement. On the other hand all techniques of fault tolerance rely on design or data redundancy to mask failure or recover from the state of failure. A direct implication of building

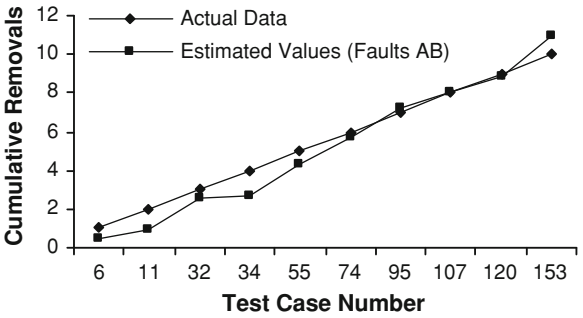


Fig. 12.24 Goodness of fit curve for fault type *AB* (discrete 3VP SRGM)

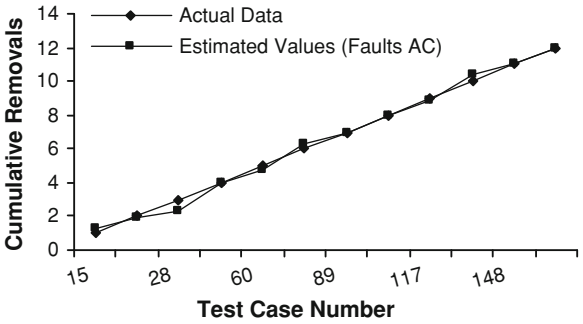


Fig. 12.25 Goodness of fit curve for fault type *AC* (discrete 3VP SRGM)

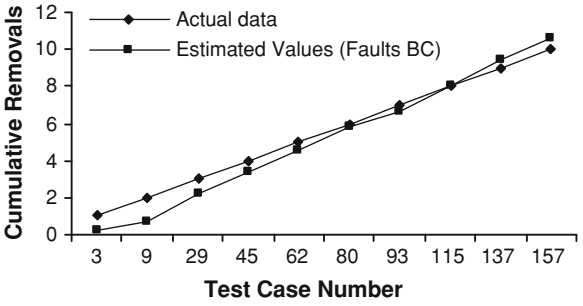


Fig. 12.26 Goodness of fit curve for fault type *BC* (discrete 3VP SRGM)

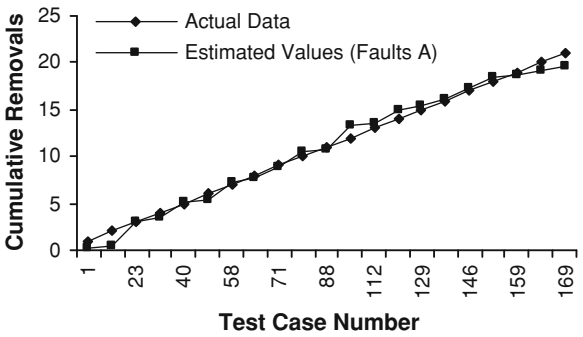


Fig. 12.27 Goodness of fit curve for fault type *A* (discrete 3VP SRGM)

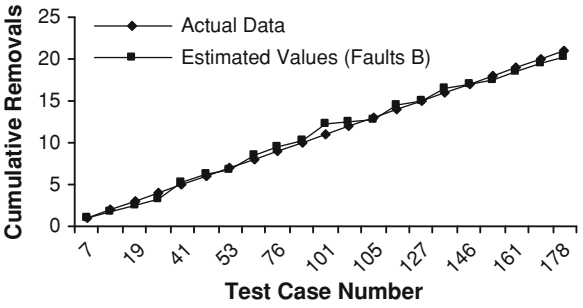


Fig. 12.28 Goodness of fit curve for fault type *B* (discrete 3VP SRGM)

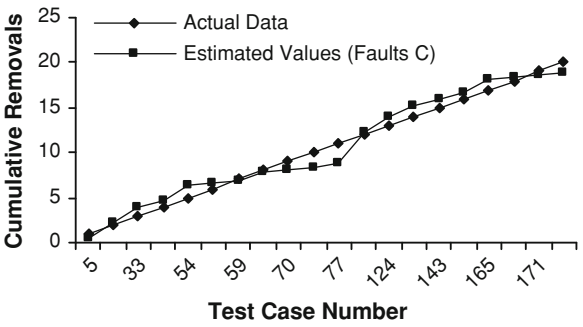


Fig. 12.29 Goodness of fit curve for fault type *C* (discrete 3VP SRGM)

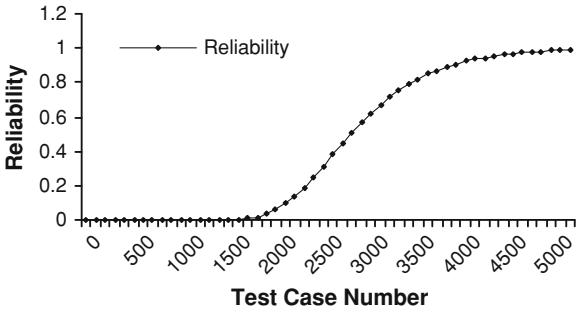


Fig. 12.30 System reliability growth (discrete 3VP SRGM)

redundancy is the additional cost. As we are all aware that with the growth of complexity and requirement of software in critical systems fault tolerance cannot be avoided. Hence it is desired that the developers must weigh the cost of fault tolerance against the cost of failure. Instead of applying ad hoc methods, use of optimization techniques is always preferred in such decision-making situations. Optimization problems of optimum selection of redundant components are widely studied by many researchers in the literature [18–22, 36]. Different optimization models were formulated each applicable to a different software system structure, ranging from a very simple configuration to more sophisticated ones. The earliest work in this field appears to be done by Belli and Jedrzejowicz [17] but it gained popularity with the work of Ashrafi and Berman [18].

12.4.1 Optimization Models for Selection of Programs for Software Performing One Function with One Program

The first problem [18] addressed by the authors is that different organizations purchase software programs to perform their different functions where each program can be weighted according to the usage frequency of the function it performs. For performing each function several programs may be available in market with known market cost and reliability. The optimization problems under consideration maximize the average reliability of a software package consisting of several programs to perform distinct functions by purchasing the best reliable programs in the market such that the total cost of package remains within the available budget. It can be noted here that the assumption of using the ready programs available in the market to make the package implies the use of COTS (*Commercial Off-the-shelf*) programs. For the COTS programs it is possible to have information on reliability and cost. Hence the models are applicable only to those software packages that are designed using COTS products. As COTS programs/modules are often developed by completely different groups, tools and in different environment, failure independence of these programs is more acceptable. The authors have formulated two types of models one which does not consider maintaining redundancy for performing each function and the other which maintains redundancy under budget limits.

Notation

K Number of functions the software package is required to perform
F_k Frequency of use of function k, $k = 1, 2, \ldots, K$
m_k Number of programs available for function k
R_{kj} Reliability of program j, which performs function k
X_{kj} are indicator variable $= \begin{cases} 1 & \text{program is selected to perform function} \\ 0 & \text{otherwise} \end{cases}$
$\overline{R}$ Average reliability of the software package
c_{kj} Cost of developing program j that performs function k
B Available budget

Assumption

1. Each program has a distinct, known reliability and cost.
2. The budget is limited.
3. Usage frequency for each function is known, viz, provided by the user.

12.4.1.1 Model Without Redundancy

The software package performs several functions but due to financial limitations and/or non-critical nature of the functions, keeping multiple programs that are functionally equivalent is not possible. The objective is to select one program for

each function such that the average reliability is maximized and the cost of purchasing programs remains within the budget. Hence the problem of maximizing average reliability by choosing the optimal set of programs is formulated as

$$\text{Maximize} \quad \overline{R} = \sum_{k=1}^{K} F_k R_k \tag{12.4.1}$$

$$\text{Subject to} \sum_{j=1}^{m_k} X_{kj} = 1,\ k = 1, \ldots, K \tag{12.4.2}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{m_k} X_{kj} C_{kj} \leq B \tag{12.4.3}$$

$$X_{kj} = 0, 1, \text{ for } k = 1, 2, \ldots, K \text{ and } j = 1, 2, \ldots, m_k \tag{P1}$$

$$R_k \equiv \sum_{j=1}^{m_k} X_{kj} R_{kj} \tag{12.4.4}$$

The objective function of the above problem reflects that the average reliability of the software package is maximized which is a weighted sum of the reliability of the K functions; reliability of each program is multiplied by the usage frequency of the corresponding function. The constraint set ensures that exactly one program is selected for each function and the total expenditure does not exceed the budget. Since the functions are required for entirely different purposes they should be considered not as a series of functions but as a set of s-independent functions.

Maximizing the average reliability, $\overline{R}$, is equivalent to minimizing the average failure rate $\overline{Z}$,

$$\overline{Z} \equiv (N/H) \sum_{k=1}^{K} F_k (1 - R_k) \tag{12.4.5}$$

where

H is the operation time,

N the number of runs during H and $1 - R_k$ the probability of failure for function k.

Using $\sum_{k=1}^{K} F_k = 1$ Eq. (12.4.5) can be rewritten as

$$\overline{Z} = (N/H)\left(1 - \sum_{k=1}^{K} F_k R_k\right) = (N/H)\left(1 - \overline{R}\right)$$

The problem (P1) is an integer programming problem and can be solved using software packages such as LINGO, LINDO, Mathematica, etc. The problem has sum of m_k; $k = 1,2,\ldots, K$ variables and $K + 1$ constraints. The authors have also proposed a Lagrangian Relaxation algorithm to solve the problem if K and m_k are large. However, now a days the professional versions of these software are available which can solve problems of very high dimensions.

Application 12.1

A software with two functions is required to be built with a budget of \$12. Four programs are available for function 1 with reliability and cost $R_{11} = 0.90,\ R_{12} = 0.80,\ R_{13} = 0.85,\ R_{14} = 0.95, C_{11} = \$6,\ C_{12} = \$4,\ C_{13} = \$5,\ C_{14} = \$8.$ For function 2, 3 alternatives are available with specifications $R_{21} = 0.70,\ R_{22} = 0.80,\ R_{23} = 0.85, C_{21} = \$2,\ C_{22} = \$4,\ C_{23} = \$6.$ The weight assigned to the usage frequency of functions 1 and 2 is $F_1 = 0.75,\ F_2 = 0.25.$

Solving the above application according to the above formulation selects program 4 for function 1 and program 2 for function 2. The optimal level of reliability is 0.9125 at the cost of \$12.

12.4.1.2 Model with Redundancy

The previous model denies the selection of more than one component for performing a specific function. Hence such software does not allow fault tolerance. Assuming the recovery block scheme of fault tolerance the problem (P1) can be reformulated as follows

$$\begin{aligned} &\text{Maximize} \quad \overline{R} = \sum_{k=1}^{K} F_k R_k \\ &\text{Subject to} \quad \sum_{j=1}^{m_k} X_{kj} \geq 1,\ k = 1, \ldots, K \\ &\qquad \sum_{k=1}^{K} \sum_{j=1}^{m_k} X_{kj} C_{kj} \leq B \end{aligned} \tag{12.4.6}$$

$$X_{kj} = 0, 1, \text{ for } k = 1, 2, \ldots, K \text{ and } j = 1, 2, \ldots, m_k \tag{P2}$$

$$R_k \equiv 1 - \prod_{j=1}^{m_k} \left(1 - R_{kj}\right)^{X_{kj}} \tag{12.4.7}$$

In contrast to (12.4.2), (12.4.6) allows selection of more than one program for any particular function. Here it is assumed that the cost of performing the acceptance test for each program is negligible compared to the purchase cost. A failure state occurs only when all alternative programs for any function give an incorrect output. Thus R_k is the probability that at least one of the programs

selected for the function k is working. This probability is hence same as that for a parallel system. The problem (P2) can be rewritten as a minimization of failure rate using (12.4.7) with the objective function

$$\text{Minimize} \sum_{k=1}^{K} F_k \prod_{j=1}^{m_k} \left(1 - R_{kj}\right)^{X_{kj}} \tag{12.4.8}$$

Like problem (P1), (P2) can also be solved using any integer-programming package. However, the authors have proposed a dynamic programming algorithm to solve the problem, as due to its nonlinear nature it cannot be solved with Lagrangian Relaxation algorithm.

Application 12.2

For the same data as in Application 12.1, the formulation (P2) gives the same solution as given by (P1) as the solution exhausts the whole budget i.e. \$12. Now if the budget is increased to \$14, an increase in budget by \$2 allows redundancy. Program 4 is selected for function 1 and programs 1 and 2 for function 2. Redundancy of program in function 2 improves reliability from 0.9125 to 0.9475, i.e. approximately by 3.5%.

In another research Berman and Ashrafi [19] formulated the optimum component selection problem for a different system structure. Here instead of assuming that software is capable of performing different functions, each being performed by specific program with known reliability and cost, they considered the system architecture where software is required to perform one or more functions. Each function is performed by executing a program, where each program consists of a series of modules which, upon sequential execution, performs the function. Each module may be called by more than one program. The optimal redundancy level is to be determined at the modular level.

12.4.2 Optimization Models for Selection of Programs for Software Performing Each Function with a Set of Modules

Four types of optimization problems were discussed applicable to the different system structures. First two models consider software performing only one function with and without maintaining redundancy at modular level and later two models consider more than one function software, again with and without redundancy.

Notation

K Number of functions the software package is required to perform
n Number of modules in the software
F_k Frequency of use of function k, $k = 1, 2, \ldots, K$
m_i Number of versions available for module i

R_{ij} Reliability of version j, for module i

X_{ij} are indicator variable $= \begin{cases} 1 & \text{Version } j \text{ is selected for module } i \\ 0 & \text{otherwise} \end{cases}$

R_i Estimated reliability of module i

$\overline{R}$ Estimated reliability of the software

c_{ij} Cost of developing version j for module i

B Available budget

S_k Set of modules corresponding to program k

Assumption

1. Software is developed using modular programming
2. Functionally equivalent versions developed independently are available for each module, with known estimated reliability and cost.
3. The budget is limited.
4. Usage frequency for each function is known, viz, provided by the user.

12.4.2.1 Optimization Models for One Function Software

These models consider software performing only one major function by executing a set of modules sequentially. Alternative versions are available for different modules.

Model 1 Without Redundancy

Due to the limitation of budget or non-critical nature of the software operation redundancy of software modules is not allowed, hence the problem can be formulated as follows

$$\text{Maximize} \quad R = \prod_{i=1}^{n} R_i \tag{12.4.9}$$

$$\text{Subject to} \quad \sum_{j=1}^{m_i} X_{ij} = 1,\ i = 1, \ldots, n \tag{12.4.10}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} X_{ij} C_{ij} \leq B$$

$$X_{ij} = 0, 1 \quad i = 1, \ldots, n \quad j = 1, \ldots, m_i \tag{P3}$$

where

$$R_i = \sum_{j=1}^{m_i} X_{ij} R_{ij} \tag{12.4.11}$$

The objective function of (P3) reflects that the modules are executed sequentially. The constraint set ensures that exactly one version is selected for each module and total expenditures will not exceed B. This problem is a nonlinear integer-programming problem. Authors suggested a branch and bound approach to solve the problem, although it can also be solved using any nonlinear integer-programming software package.

Application 12.3

Consider single function software consisting of a set of three modules (say A, B, and C). The number of alternatives available for modules A, B, and C is 3, 3, and 2. The reliability and cost of the modules are $R_{11} = 0.90$, $R_{12} = 0.80$, R_{13}=0.85, $R_{21} = 0.95$, $R_{22} = 0.80$, $R_{31} = 0.98, R_{32} = 0.94, C_{11} = \$3$, $C_{12} = \$1$, $C_{13} = \$2$, $C_{21} = \$3$, $C_{22} = \$2, C_{23} = \$1, C_{31} = \$3, C_{32} = \$2$. The budget is \$6.

The optimal solution obtained selects the versions 2, 1, and 2 respectively for modules A, B, and C. The reliability level achieved is 0.714 with the total expenditure of \$6.

Model 2 With Redundancy

This model considers software, which performs a more critical function, whose failure can be severe. Fault tolerance is built by keeping redundant versions for modules. Hence the problem (P3) is restated as

$$\begin{aligned} &\text{Maximize} \quad R = \prod_{i=1}^{n} R_i \\ &\text{Subject to} \quad \sum_{j=1}^{m_i} X_{ij} \geq 1,\ i = 1, \ldots, n \\ &\qquad \sum_{i=1}^{n} \sum_{j=1}^{m_i} X_{ij} C_{ij} \leq B \end{aligned} \tag{2.3.12}$$

$$X_{ij} = 0, 1 \quad i = 1, \ldots, n \quad j = 1, \ldots, m_i \tag{P4}$$

where

$$R_i = 1 - \prod_{j=1}^{m_i} \left(1 - R_{ij}\right)^{X_{ij}} \tag{12.3.13}$$

The reliability of modules i is defined as the probability that at least one of the m_i versions is performing correctly. The constraint set guarantees that for each module i at least one version will be selected. Again this problem is also a non-linear integer-programming problem. Authors have suggested a Dynamic programming algorithm for solving the problem. Since this problem can be solved using software packages we avoid discussing the algorithm.

Application 12.4

Consider the same data as in Application 12.3 with a budget of \$10, the optimal solution found is $X_{12} = X_{13} = 1$; $X_{21} = X_{23} = 1$; $X_{31} = 1$. The overall reliability achieved is 0.9359 and cost \$10. Reliability is 22.19% more as compared to the one obtained in without redundancy formulation with an increased cost of \$4.

12.4.2.2 Optimization Models for Multiple (K) Function Software

These models consider software system consisting of several programs each consisting of a set of modules and performing a specific function. Programs can call any module and alternative versions are available for different modules.

Model 3 Without Redundancy

Due to the limitation of budget or non-critical nature of the software operation redundancy of software modules is not allowed, hence the problem of determining the optimal set of modules maximizing reliability within the budget constraint can be formulated as follows

$$\text{Maximize} \quad R = \sum_{k=1}^{K} F_k \prod_{i \in S_k} R_i$$

$$\text{Subject to} \quad \sum_{j=1}^{m_i} X_{ij} = 1, \quad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} X_{ij} C_{ij} \leq B$$

$$X_{ij} = 0, 1 \quad i = 1, \ldots, n \quad j = 1, \ldots, m_i \tag{P5}$$

where $R_i = \sum_{j=1}^{m_i} X_{ij} R_{ij}$.

The objective function (P5) reflects that the modules in any program are executed sequentially each having usage frequency F_k. The constraint set is same as in (P3). Authors suggested a branch and bound approach as well as the use of any nonlinear integer-programming software package to solve the problem.

Application 12.5

Consider two-function software consisting of a set of three modules (say A, B, and C). Two alternatives are available for each module. Function one is performed by executing program one consisting of modules A and B sequentially. While another program consisting of modules B and C performs the function 2 on sequential execution of its modules. The usage frequency of functions 1 and 2 is $F_1 = 0.70$ and $F_2 = 0.30$. The reliability and cost of the modules are $R_{11} = 0.80$, $R_{12} = 0.85$, $R_{21} = 0.70$, $R_{22} = 0.90$, $R_{31} = 0.95$, $R_{32} = 0.90$, $C_{11} = \$2$, $C_{12} = \$3$, $C_{21} = \$1$, $C_{22} = \$3$, $C_{31} = \$4$, $C_{32} = \$3$. The budget is \$8.

The optimal solution is (X_{11}, X_{22}, X_{32}) at the cost of \$8 with the achieved level of reliability 0.747. If the budget is \$10 then the optimal solution changes to (X_{12}, X_{22}, X_{31}) with the total expenditure of \$10 and reliability level 0.792.

Model 4 With Redundancy

Permitting redundancy in (P5) the problem is restated as

$$\begin{aligned} &\text{Maximize} \quad R = \sum_{k=1}^{K} F_k \prod_{i \in S_k} R_i \\ &\text{Subject to} \quad \sum_{j=1}^{m_i} X_{ij} \geq 1, \quad i = 1, \ldots, n \\ &\qquad \sum_{i=1}^{n} \sum_{j=1}^{m_i} X_{ij} C_{ij} \leq B \\ &X_{ij} = 0, 1 \quad i = 1, \ldots, n \quad j = 1, \ldots, m_i \end{aligned} \tag{12.4.14}$$

(P6)

where $R_i = 1 - \prod_{j=1}^{m_i} \left(1 - R_{ij}\right)^{X_{ij}}$.

Again the problem is nonlinear integer-programming problem and can be solved with the help of software packages.

Application 12.6

Considering the same data as in Application 12.5 with a budget of \$9 the optimal solution is (X_{11},X_{21}, X_{22}, X_{32}) with the achieved level of reliability 0.8052. Note that the achieved level of reliability is even better than the case of Application 12.5 when the budget is \$10.

12.4.3 Optimization Models for Recovery Blocks

The optimization models discussed in the previous sections do not consider any of the fault tolerance schemes such as recovery block or NVP. They merely

consider the programs consisting of set of modules, which on sequential execution perform the function. Berman and Kumar [22] studied the problem of optimum selection of component for the recovery blocks for the first time. Specifically they have formulated optimization problems for two types of recovery blocks namely—Independent and CRB schemes.

Notation

n	Number of versions in the recovery block
$P(.)$	Probability of the event (.)
p_i	Failure probability of version i
C_i	Cost of version i
t	Maximum time available for a version to submit output to the testing segment
t_1	Probability that the testing segment can not perform successful recovery of the input state
t_2	Probability that the testing segment rejects a correct result
t_3	Probability that the testing segment accepts an incorrect result
B	Available budget
R_n	Reliability of a recovery block with n versions
RC_n	Reliability of a consensus recovery block (CRB) with n versions
Z_j	Binary variable $= \begin{cases} 1 & \text{if version } j \text{ is included in the software} \\ 0 & \text{otherwise} \end{cases}$
$Z_{i,j}$	Binary variable $= \begin{cases} 1 & \text{if version } i \text{ is placed at position } j \text{ in the software} \\ 0 & \text{otherwise} \end{cases}$
R_j	Reliability for the partial solution $\{\overline{Z}_1, \overline{Z}_2, \ldots, \overline{Z}_j\}$
$1/\lambda_i$	Mean execution time for version i

12.4.3.1 Independent Recovery Block

Different types of errors that can result in failure of a recovery block are

1. A version produces correct result, but the testing segment labels it incorrect
2. A version produces incorrect result, but the testing segment labels it correct
3. A testing segment cannot perform successful recovery upon failure of a version

To compute the reliability of a recovery block based on the failure modes observed two types of events are defined. Let

Y_i: event that version i produces a correct result and testing segment accepts it as correct

X_j: event that either version i produces an incorrect result or produces a correct result and the testing segment rejects it; in both cases the testing segment performs a successful recovery of the input states.

The probabilities corresponding to the above two events are given as

$$P(Y_i) = (1 - p_i)(1 - t_2)$$

$$P(X_i) = (1 - t_1)[p_i(1 - t_3) + (1 - p_i)t_2]$$

Now the reliability of a recovery block scheme with a single version 1, R_1, is defined as

$$R_1 = P(Y_1)$$

In general reliability of a recovery block with n versions is

$$R_n = P(Y_1) + \sum_{i=2}^{n} \left[\prod_{k=2}^{i-1} P(X_k)\right] P(Y_i), \quad n \geq 2 \tag{12.4.15}$$

Recursively,

$$R_n = R_{n-1} + \left[\prod_{k=1}^{n-1} P(X_k)\right] P(Y_n), \quad n \geq 2 \tag{12.4.16}$$

To attain the largest possible reliability of a recovery block the different versions are to be installed in the order from smallest to the largest based on failure probabilities. This has been proved by Berman and Kumar [22] stating

Theorem 12.1 *For a recovery block scheme with n independent versions, the list ordered from smallest to the largest based on failure probabilities is at least as reliable as any other list of the n versions.*

Proof Let R_n^1 and R_n^2 be the reliability of recovery block, respectively, for list 1 and 2 where

List 1 1, 2, 3, …j, $j + 1$, …, n

List 2 1, 2, 3, … $j - 1$, $j + 1$, j, $j + 2$, …, n

This is sufficient to prove that if $P_1 \leq P_2 \leq P_3 \leq$, …, P_n then $R_n^1 \geq R_n^2$ □

From Eq. (12.4.15) R_n^1 and R_n^2 are given by

$$\begin{aligned} R_n^1 = P(Y_1) + \sum_{i=1}^{j-2} \left[\prod_{k=1}^{i} P(X_k)\right] P(Y_{i+1}) + \left[\prod_{k=1}^{j-1} P(X_k)\right] P(Y_j) \\ + \left[\prod_{k=1}^{j} P(X_k)\right] P(Y_{j+1}) + \sum_{i=j+1}^{n-1} \left[\prod_{k=1}^{i} P(X_i)\right] P(Y_{i+1}) \end{aligned} \tag{12.4.17}$$

$$\begin{aligned} R_n^2 = P(Y_1) + \sum_{i=1}^{j-2} \left[\prod_{k=1}^{i} P(X_k)\right] P(Y_{i+1}) + \left[\prod_{k=1}^{j-1} P(X_k)\right] P(Y_{j+1}) \\ + \left[\prod_{k=1}^{j-1} P(X_k)\right] P(X_{j+1}) P(Y_j) + \sum_{i=j+1}^{n-1} \left[\prod_{k=1}^{i} P(X_i)\right] P(Y_{i+1}) \end{aligned} \tag{12.4.18}$$

$$R_n^1 - R_n^2 \geq 0$$

$$\Leftrightarrow P(Y_j) + P(X_j)P(Y_{j+1}) - P(Y_{j+1}) - P(X_{j+1})P(Y_j) \geq 0 \tag{12.4.19}$$

Substituting the values of $P(Y_j)$, $P(Y_{j+1})$, $P(X_j)$ and $P(X_{j+1})$ we get

$$(1 - t_2)(P_{j+1} - P_j) - (1 - t_1)(1 - t_2)(1 - t_3)(P_{j+1} - P_j) \geq 0 \tag{12.4.20}$$

above inequality is true only if $P_{j+1} - P_j \geq 0$, hence $R_n^1 \geq R_n^2$.

Now the problem of maximizing reliability by choosing an optimal set of versions subject to a budget constraint is

$$\text{Maximize} \quad R_n = \sum_{i=1}^{n} Z_i \left(\prod_{k=1}^{i-1} P(X_k)^{Z_k} \right) P(Y_i)^{Z_i} \tag{12.4.21}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} C_i Z_i \leq B$$

$$Z_i = 0, 1 \quad i = 1, \ldots, n \tag{P7}$$

Equation (12.4.21) defines the reliability of a recovery block scheme chosen from n versions corresponding to a solution$\{Z_1, Z_2, \ldots, Z_j\}$. The constraint ensures the budget restriction. Berman and Kumar [22] developed a branch and bound algorithm to solve the problem, although they have also suggested to use any mathematical programming software package to solve the problem. Hence here we avoid discussing their algorithm and use software package LINGO to solve the problem. □

Application 12.7

Consider a recovery block scheme with four versions. The budget is $B = \$22$. The probability of failure and cost of the four versions are $p_1 = 0.1, p_2 = 0.2, p_3 = 0.3, p_4 = 0.4, c_1 = 9, c_2 = 7, c_3 = 8, c_4 = 6$ and $t_1 = 0.01, t_2 = 0.05$ and $t_3 = 0.3$.

From these data we have $P(X_1) = 0.1425$, $P(X_2) = 0.2356$, $P(X_3) = 0.3286$, $P(X_4) = 0.4217$, $P(X_4) = 0.4217$, $P(Y_1) = 0.855$, $P(Y_2) = 0.76$, $P(Y_3) = 0.665$, $P(Y_4) = 0.57$. The optimal solution found is $\{\overline{Z}_1 = 1, \overline{Z}_2 = 1, \overline{Z}_3 = 0, \overline{Z}_4 = 1\}$. The corresponding objective function value is 0.9824 and the total cost = \$22.

12.4.3.2 Consensus Recovery Block

Constitutes of a CRB are n independent versions of a program, an acceptance test and voting procedure. The versions are ranked, based on their failure procedure. Upon invocation of CRB, all versions are executed simultaneously and submit their outputs to a voting procedure. If the outputs of two or more versions are in

agreement that output is designated as correct. Otherwise the next stage is entered. At this stage the best version is examined by an acceptance test. If the output is accepted, it is treated as correct. However, if the output is not accepted, the next best version is subject to testing. This process continues until acceptable output is found or all the n outputs are exhausted.

Define the following probabilities

$$P(G_n) = P(2 \text{ or more outputs agree})$$

$$P(G_c) = P(\text{recurring output is correct})$$

$$P(D_n) = P(\text{all the outputs are different})$$

Reliability of CRB is hence given as

$$RC_n = P(G_n)P(G_c) + P(D_n)R_n$$

$$RC_n = (1 - P(D_n))P(G_c) + P(D_n)R_n \tag{12.4.22}$$

where R_n is the reliability of a recovery block scheme with n versions as given by (12.4.15) or (12.4.16). For CRB $t_1 = 0$ hence the probabilities of events Y_i and X_i are

$$P(Y_i) = (1 - p_i)(1 - t_2)) \tag{12.4.23}$$

$$P(X_i) = p_i(1 - t_3) + (1 - p_i)t_2 \tag{12.4.24}$$

Here $P(D_n)$ is the probability that at least $(n - 1)$ versions of the n versions fail. Therefore

$$P(D_n) = \sum_{i=1}^{n} \frac{1}{p_i} \left[\prod_{k=1}^{n} p_k \right] (1 - p_i) + \prod_{i=1}^{n} p_i = P(D_{n-1})p_n + \left(\prod_{i=1}^{n-1} p_i \right)(1 - p_n) \tag{12.4.25}$$

To simplify it is assumed that $P(G_c) = 1$.

$$RC_n = 1 - P(D_n) + P(D_n)R_n = 1 + P(D_n)[R_n - 1]$$

$$\text{hence } RC_n = 1 + \left[\sum_{i=1}^{n} \frac{1}{p_i} \left[\prod_{k=1}^{n} p_k \right] (1 - p_i) + \prod_{i=1}^{n} p_i \right] \left[\sum_{i=1}^{n} \left[\prod_{k=1}^{i-1} P(X_k) \right] P(Y_i) - 1 \right] \tag{12.4.26}$$

Improvement in reliability of CRB scheme over recovery block is

$$\begin{aligned} RC_n - R_n &= 1 - P(D_n) + P(D_n)R_n - R_n \\ &= (1 - P(D_n))(1 - R_n) \end{aligned} \tag{12.4.27}$$

Similar to the case of independent recovery blocks reliability of a CRB is largest if the n versions are arranged in the order of their reliability. It is proved in the following theorem.

Theorem 12.2 *For a CRB scheme with n versions, the list of versions ordered from smallest to largest based on failure probability is more reliable than any other list of versions.*

This theorem can be proved on the similar lines of the theorem established for the case of independent recovery blocks.

Now the problem of maximizing reliability by choosing an optimal set of versions subject to a budget constraint is

$$\text{Maximize} \quad RC_n = 1 + \left\{ \begin{array}{l} \left[\sum_{i=1}^{n} \frac{1}{p_i^{z_i}} \left[\prod_{k=1}^{n} p_i^{z_i}\right](1 - p_i^{z_i}) + \prod_{i=1}^{n} p_i^{z_i}\right] \\ \left[\sum_{i=1}^{n} z_i \left[\prod_{k=1}^{i-1} P(X_k)^{z_k}\right] P(Y_i)^{z_i} - 1\right] \end{array} \right\} \tag{12.4.28}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} C_i Z_i \leq B$$

$$Z_i = 0, 1 \quad i = 1, \ldots, n \tag{P8}$$

Equation (12.4.28) defines the reliability of a CRB scheme chosen from n versions corresponding to a solution$\{Z_1, Z_2, \ldots, Z_j\}$. The constraint ensures the budget restriction. Berman and Kumar [22] developed a branch and bound algorithm to solve the problem, although they have also suggested to use any mathematical programming software package to solve the problem. We use software package LINGO to solve the problem.

Application 12.8

Consider a CRB scheme with four versions and the data same as in Application 12.7. Again $t_1 = 0$, $t_2 = 0.05$, and $t_3 = 0.01$.$P(X_1) = 0.144$, $P(X_2) = 0.238$, $P(X_3) = 0.332$, $P(X_4) = 0.426$, $P(Y_1) = 0.855$, $P(Y_2) = 0.76$, $P(Y_3) = 0.665$, $P(Y_4) = 0.57$. The optimal solution found is $\{\overline{Z}_1 = 1, \overline{Z}_2 = 1, \overline{Z}_3 = 0, \overline{Z}_4 = 1\}$. The corresponding objective function value is 0.9980 and the total cost = \$22. Improvement in the reliability of CRB over independent recovery block is 0.0156 (= 0.9980−0.9824).

12.4.3.3 Independent Recovery Block with Exponential Execution Time

In the previous models it is assumed that all the versions of a block submit the output to the acceptance test. However, some versions may enter into infinite loop

and may not submit any output at all. In this model it is assumed that the execution time of each version follows an exponential distribution with rate λ_i, for version i. Kumar [20] defined the errors that can result in failure of a recovery block with exponential execution time

1. A version produces incorrect result (i.e. completes execution by time t) and the testing segment labels it incorrect.
2. A version fails to complete execution by time t.
3. A version produces correct result, but the testing segment labels it incorrect.
4. A version produces incorrect result, but the testing segment labels it correct.
5. The testing segment cannot perform successful recovery upon failure of a version.

To compute the reliability of a recovery block based on the failure modes observed two types of events are defined. Let

Y_i: Event that version i produces a correct result by time t and testing segment accepts the correct result.

X_j: Event that (1) the version i produces an incorrect result and the testing segment rejects it or that the version i produces a correct result and testing segment rejects or (2) the version does not complete execution by time t. In each case the testing segment performs a successful recovery of the input states.

The probabilities corresponding to the above two events are given as

$$P(Y_i) = (1 - p_i)(1 - t_2)(1 - \exp(-\lambda_i t))$$

$$P(X_i) = (1 - t_1)[\exp(-\lambda_i t) + (1 - \exp(-\lambda_i t))(p_i(1 - t_3) + (1 - p_i)t_2)]$$

Now the reliability of a recovery block scheme with a single version 1, R_1, is defined as

$$R_1 = P(Y_1)$$

In general reliability of a recovery block with n versions having exponential execution times is

$$R_n = P(Y_1) + \sum_{i=1}^{n-1} \left[\prod_{k=1}^{i} P(X_k) \right] P(Y_{i+1}), \quad n \geq 2 \tag{12.4.29}$$

Recursively,

$$R_n = R_{n-1} + \left[\prod_{k=1}^{n-1} P(X_k) \right] P(Y_n), \quad n \geq 2 \tag{12.4.30}$$

To attain the largest possible reliability of a recovery block the different versions are to be installed in the order from smallest to the largest based on failure probabilities. This has been proved by Kumar [20] that for a recovery block scheme with exponential execution time of versions the optimal sequence is based on the value V_i, where

$$V_i = \frac{P(Y_i)}{(1 - P(X_i))}$$

Theorem 12.3 *For a recovery block scheme with n independent versions and exponential execution time, the list ordered from largest to smallest based on V_i is at least as reliable as any other list of n versions.*

Proof: Let R_n^1 and R_n^2 be the reliability of recovery block, respectively, for list 1 and 2 where

List 1 1, 2, 3, …, $i - 1$, i, $i + 1$, …, n
List 2 1, 2, 3, …, $i - 1$, $i + 1$, i, $i + 2$, …, n □

This is sufficient to prove that, $R^1 \geq R^2$ iff $V_i \geq V_{i+1}$.
The expression for R^1 and R^2 iss

$$R^1 = R_1 + \sum_{j=1}^{i-2} \left[\prod_{k=1}^{j} P(X_k)\right] P\left[Y_{j+1}\right] + \left[\prod_{k=1}^{i-1} P(X_k)\right] P(Y_i) + \left[\prod_{k=1}^{i-1} P(X_k)\right] P(X_i) P(Y_{i+1}) + \sum_{j=i+1}^{n-1} \left[\prod_{k=1}^{j} P(X_k)\right] P\left(Y_{j+1}\right)$$

$$R^2 = R_1 + \sum_{j=1}^{i-2} \left[\prod_{k=1}^{j} P(X_k)\right] P\left[Y_{j+1}\right] + \left[\prod_{k=1}^{i-1} P(X_k)\right] P(Y_{i+1}) + \left[\prod_{k=1}^{i-1} P(X_k)\right] P(X_{i+1}) P(Y_i) + \sum_{j=i+1}^{n-1} \left[\prod_{k=1}^{j} P(X_k)\right] P\left(Y_{j+1}\right)$$

$$R^1 - R^2 = \prod_{k=1}^{i-1} P(X_k) P(Y_i) + \left[\prod_{k=1}^{i-1} P(X_k)\right] P(X_i) P(Y_{i+1}) - \left[\prod_{k=1}^{i-1} P(X_k)\right] P(Y_{i+1}) - \left[\prod_{k=1}^{i-1} P(X_k)\right] P(X_{i+1}) P(Y_i)$$

$$R^1 - R^2 \geq 0 \Leftrightarrow P(Y_i)(1 - P(X_{i+1})) - P(Y_{i+1})(1 - P(X_i)) \geq 0$$

$$\Leftrightarrow \frac{P(Y_i)}{\left(1 - P\left(X_j\right)\right)} \geq \frac{P(Y_{i+1})}{(1 - P(X_{i+1}))}; \quad i.e \quad V_i \geq V_{i+1}.$$

Now the problem of maximizing reliability subject to a budget constraint is

Model 1

$$\text{Maximize } R_n = \sum_{i=1}^{n} P(Y_i)Z_{i,1} + \sum_{j=1}^{n-1}\left[\prod_{k=1}^{j}\sum_{i=1}^{n} P(X_i)Z_{i,k}\right]\left[\sum_{i=1}^{n} P(Y_i)Z_{i,j+1}\right] \tag{12.4.31}$$

$$\text{Subject to } \sum_{i=1}^{n} C_i Z_{i,j} \le B \tag{P9}$$

$$\sum_{i=1}^{n} Z_{i,j} = 1, \quad j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} Z_{i,j} = 1, \quad i = 1, 2, \ldots, n$$

$$Z_{i,j} = 0, 1 \qquad i = 1, \ldots, n;\ j = 1, \ldots, n$$

Equation (12.4.21) defines the reliability of an independent recovery block scheme. The constraint ensures the budget restriction, and that each version is executed only once. The problem can be solved using any mathematical programming software package. Here we use software package LINGO to solve the problem.

Application 12.9

Consider a recovery block scheme with four versions. The budget is $B = \$22$. The probability of failure, cost and mean execution time of the four versions are $p_1 = 0.05$, $p_2 = 0.1$, $p_3 = 0.15$, $p_4 = 0.2$; $c_1 = 9$, $c_2 = 7$, $c_3 = 8$, $c_4 = 5$; $(1/\lambda_1) = 10, (1/\lambda_2) = 8$, $(1/\lambda_3) = 5, (1/\lambda_4) = 4$ and $t_1 = 0.01$, $t_2 = 0.05$ and $t_3 = 0.01$ and $t = 10$. From these data we have $P(X_1) = 0.45431$, $P(X_2) = 0.45173$, $P(X_3) = 0.41818$, $P(X_4) = 0.46838$, $P(Y_1) = 0.54046$, $P(Y_2) = 0.54225$, $P(Y_3) = 0.575$, $P(Y_4) = 0.52321$, $V_1 = 0.9919$, $V_2 = 0.9925$, $V_3 = 0.9938$, $V_4 = 0.9931$. The optimal solution found is $\{Z_{3,1} = 1,\ Z_{3,k} = 0,\ k \neq 1;\ Z_{4,2} = 1,\ Z_{4,k} = 0,\ k \neq 2;\ Z_{2,3} = 1,\ \gamma_{2,k} = 0,\ k \neq 3\}$. The corresponding objective function value is 0.9597 and the total cost = \$20.

Model 2

In this model an additional "time constraint" is introduced.

$$\text{Time Constraint} : \sum_{i=1}^{n}\sum_{j=1}^{n} (1/\lambda_i)Z_{i,j} \le T \tag{P10}$$

Time constraint guarantees that the total execution time of the recovery block is within the maximum allowed time.

Application 12.10

Consider the same data as in Application 12.10 with $T = 20$. The optimal solution found is $\{Z_{3,1} = 1, Z_{3,k} = 0, k \neq 3; Z_{4,2} = 1, Z_{4,k} = 0, k \neq 2; Z_{2,3} = 1, \gamma_{2,k} = 0, k \neq 3\}$. The corresponding objective function value is 0.9597, total cost = \$20 and $T = 17$.

12.4.4 Optimization Models for Recovery Blocks with Multiple Alternatives for Each Version Having Different Reliability

The optimal component selection problem addressed due to Kapur et al. [36] considers software built by assembling COTS component performing multiple functions. Each function is performed calling a set of modules. Modules can be assembled in a recovery block scheme to provide the fault tolerance. For performing the function of each module alternative COTS component is available in the market. Again for each alternative version multiple choices are available from the supplier with distinct reliability and cost. The version for any alternative having higher reliability has higher cost. Two models are formulated for weighted maximization of system reliability, weights being decided with respect to access frequency of functions with in the available budget.

Notation

$\overline{R}$ Estimated reliability of the software
F_l Frequency of use of function l, $l = 1, 2, \ldots L$
S_l Set of modules required for function l
R_i Estimated reliability of module i
L Number of functions the software package is required to perform
n Number of modules in the software
m_i Number of alternatives available for module i
V_{ij} Number of versions available for i
R_{ij} Reliability of alternative j, for module i
R_{ijk} Reliability of version k of alternative j, for module i
X_{ijk} Is indicator $= \begin{cases} 1 & \text{Version } k \text{ of alternative } j \text{ is selected for module } i \\ 0 & \text{otherwise} \end{cases}$
C_{ijk} Cost of version k of alternative j for module i
B Available budget
M Large number greater than 1
Y_i Is indicator $= \begin{cases} 1 & \text{if constraint } i \text{ is inactive} \\ 0 & \text{otherwise} \end{cases}$
z Number of alternatives compatible for module with respect to another module

Assumption

1. Codes written for integration of modules do not contain any bug.
2. Other than available cost-reliability versions of an alternative, existence of virtual versions is assumed having negligible reliability of 0.001 and zero cost. Existence of virtual versions allows no redundancy, in case of insufficient budget. These components are denoted by index one in the third subscript of x_{ijk}, c_{ijk} and r_{ijk}; for example, r_{ij1} is reliability of first version of alternatives j for module i, having the above property.

Model 1

The optimization problem of model 1 is

$$\text{Maximize} \quad \bar{R} = \sum_{l=1}^{L} F_l \prod_{i \in s_l} R_i$$

$$\text{Subject to} \quad \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{V_{ij}} C_{ijk} X_{ijk} \leq B \tag{P11}$$

$$R_i = 1 - \prod_{j=1}^{m_i} \left(1 - R_{ij}\right); \quad i = 1, 2, \ldots, n$$

$$R_{ij} = \sum_{k=1}^{V_{ij}} X_{ijk} R_{ijk} \quad i = 1, 2, \ldots .n; \quad j = 1, 2, \ldots m_i$$

$$\sum_{k=1}^{V_{ij}} X_{ijk} = 1, \qquad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m_i$$

$$R_i > 1 - \prod_{j=1}^{m_i} \left(1 - R_{ij1}\right)$$

Objective function maximizes software reliability through a weighted function of functional usage frequencies. Reliability of functions that are performed more frequently, consequently the modules that are invoked more frequently during use are given higher weights. The first constraint ensures the budget restriction. As it is assumed that the exception raising and control transfer programs work perfectly, a module fails if all attached alternatives fail. Hence the reliability expression is similar to parallel structure as in second constraint. Third constraint computes the reliability of the jth alternative for module i. Fourth constraint ensures that only one version will be chosen for any particular alternative, which can also be the dummy version. The last constraint ensures not all the selected alternatives for any module are dummies. This model is a 0−1 nonlinear integer-programming

Table 12.13 Data for Application 12.11

Module	Alternatives	Versions (Cost in $)		
		1	2	3
1	1	0.0	8.2	9.0
	2	0.0	7.5	9.0
	3	0.0	8.5	9.5
2	1	0.0	6.7	8.0
	2	0.0	7.9	8.2
	3	0.0	6.8	7.8
3	1	0.0	3.2	4.0
	2	0.0	3.4	4.3
4	1	0.0	5.0	6.8
	2	0.0	4.8	6.8
5	1	0.0	3.8	6.2
	2	0.0	4.2	6.0
R_{ijk}		0.001	0.85	0.95

problem. We now illustrate the model with an application solved using software package LINGO.

Application 12.11

Consider software capable of performing four functions consisting of a set of five modules. More than one alternative is available for each module with two versions available for each alternative. One virtual version is assumed to exist for each alternative indexing them with index one in the third subscript. The cost-reliability data are as summarized in the following Table 12.13.

Assume the budget is \$38. $S_1 = \{1, 2, 3, 4, 5\}$, $S_2 = \{1, 2, 5\}$, $S_3 = \{1, 2, 3, 5\}$, $S_4 = \{1, 2, 4, 5\}$, $F_1 = 0.5$, $F_2 = 0.1$, $F_3 = 0.2$, $F_4 = 0.2$. The optimal solution obtained is $x_{111} = x_{123} = x_{131} = 1$; $x_{211} = x_{221} = x_{233} = 1$; $x_{313} = x_{321} = 1$; $x_{413} = x_{421} = 1$; $x_{513} = x_{522} = x_{531} = 1$. From the solution it can be seen that only one alternative is chosen for first to fourth modules. Redundancy is allowed only in the fifth module. The redundant component for fifth module i.e. X_{522} (one having lesser reliability) does not have the highest reliability among the available versions, this is due to budget limitation. For all other alternatives, the virtual version is chosen in the solution. The achieved level of reliability is 0.8344 at the cost of \$38.

Model 2

A very common problem associated to the use of COTS component is that some of the alternatives available for one module may not be compatible with some alternatives of another module. This issue must be considered with formulating the optimum component selection problem. None of the models discussed so far accounts it. The model 2 formulated due to Kapur et al. [36] accounts the compatibility of the module in forming the model. The additional constraints included in the optimization problem of model 2 are

$$x_{gsq} - x_{hu_t c} \leq My_t \; q = 2, \ldots, V_{gs}, \; c = 2, \ldots, V_{hu_t}, s = 1, \ldots, m_g \tag{12.4.32}$$

$$\sum_{t=1}^{d} y_t = d - 1 \tag{12.4.33}$$

where

$$d = \left(V_{gs} - 1\right)\left(V_{hu_t} - 1\right) \tag{P12}$$

The constraints (12.4.32) and (12.4.33) make use of binary variable y_t to choose one pair of alternatives from among different alternative pairs of modules. If more than one alternative compatible component is to be chosen for redundancy, constraint (12.4.33) can be relaxed as follows,

$$\sum_{t=1}^{d} y_t \leq d - 1$$

This model is also a 0−1 nonlinear integer-programming problem. We now illustrate the model with an application solved using software package LINGO.

Application 12.12

Consider the data same as in Application 12.11 and that first alternative of module 1 is compatible with second and third alternatives of module 2. The first alternative of second module is compatible with second alternative of module 3. Lastly the first alternative of fourth module is compatible with second and third alternatives of module 5. The solution obtained with the budget of \$38 is $x_{111} = x_{123} = x_{131} = 1$; $x_{211} = x_{221} = x_{233} = 1$; $x_{313} = x_{321} = 1$; $x_{411} = x_{423} = 1$; $x_{513} = x_{522} = x_{531} = 1$; It is observed that due to the compatibility condition, first alternative of module 5 is not chosen as in case of Application 12.11. The system reliability is 0.8343.

Exercises

1. What SRE techniques can be used to minimize the system failure during field use?
2. Software used to control and operate critical system applications have very high reliability requirement. Why one needs to build fault tolerance in such systems even though reliability of such software can be assured at very high level by the use of scientific testing techniques and continuing testing for long duration. Comment.
3. Write short note on the following techniques of fault tolerance.
   a. Design diversity
   b. Data diversity
   c. Environment diversity.

4. What are the two important design diversity techniques used in software industry? Explain.
5. What is a hybrid design diversity technique?
6. Classify the faults in NVP system. Give a pictorial representation of these faults in a 3VP system.
7. What is the COTS technology?
8. Explain the optimum component selection problem related to the development of COTS product with the help of a diagram.
9. A software is supposed to perform three functions; different alternative software programs are available for each function with varying cost and reliability as given in the following table. The frequency of use of functions 1, 2, and 3 is 0.2, 0.5, and, 0.3, respectively. If a budget of $34 is available what will be the optimum structure of the software assuming redundancy is not required.

Alternative	Program 1		Program 2		Program 3	
	Reliability	Cost ($)	Reliability	Cost ($)	Reliability	Cost ($)
1	0.85	9	0.9	4	0.78	8
2	0.86	13	0.93	8	0.82	12
3	0.9	20	0.95	10	0.84	15
4	0.92	24	–	–	–	–

10. Assume that redundancy is allowed in exercise 9. what will be the optimum structure of the software if budget is kept same? Also determine the optimum solution for a budget of $50. Give the system reliability in each case.
11. Determine the optimal solution of Application 12.12, if the budget is changed from $38 to $68, keeping all other data same. What will be the level of reliability achieved?

References

1. Avizienis A, Kelly JPJ (1984) Fault tolerance by design diversity: concepts and experiments. IEEE Computer 17(8):67–80
2. Ammann PE, Knight JC (1988) Data diversity: an approach to software fault tolerance. IEEE Trans Comput. 37(4):418–425
3. Jalote P, Huang Y, Kintala C (1995) A framework for understanding and handling transient software failures. In: Proceedings of the 2nd ISSAT International Conference on Reliability and Quality in Design, Orlando, pp 231–237
4. Adams E (1994) Optimizing preventive service of the software products. IBM J R&D 28(1):2–14
5. Lee I, Iyer RK (1995) Software dependability in the tandem GUARDIAN system. IEEE Trans Softw Eng 21(5):455–467
6. Avizienis A (1975) Fault-tolerance and fault-intolerance: complementary approaches to reliable computing. Presented at international conference on reliable software, Los Angeles, California

7. Randell B (1975) System structure for software fault tolerance. IEEE Trans Softw Eng SE-1(2):220–232
8. Chen L, Avizienis A (1978) N-version programming: a fault tolerance approach to the reliable software. In: Proceedings of the 8th international symposium fault-tolerant computing, Toulouse, pp 3–9
9. Leung YW (1995) Maximum likelihood voting for fault tolerant software with finite output spaces. IEEE Trans Reliability 44(3):419–426
10. Horning JJ, Lauer HC, Melliar PM, Randell B (1974) A program structure for error detection and recovery. Lect Notes Comput Sci 16:177–193
11. Nicola VF, Goyal A (1990) Modeling of correlated failures and community error recovery in multi-version software. IEEE Trans Softw Eng 16(3):350–359
12. Yau SS, Cheung RC (1975) Design of self-checking software. In: Proceedings of the international conference on reliable software, IEEE Computer Society Press, Los Angeles pp 450–457
13. Hecht M, Agron J, Hochhauser S (1989) A distributed fault tolerant architecture for nuclear reactor control and safety functions. In: Proceedings of the real-time system symposium, Santa Monica, pp 214–221
14. Scott RK, Gault JW, McAllister DF (1985) Fault tolerant software reliability modeling. IEEE Trans Softw Eng 13(5):582–592
15. Scott RK, Gault JW, McAllister DF (1987) Fault-tolerant reliability modeling. IEEE Trans Softw Eng SE-13(5):582–592
16. Lyu MR (1995) Software fault tolerance. Wiley, New York
17. Belli F, Jedrzejowicz P (1990) Fault-tolerant programs and their reliability. IEEE Trans Reliability 29(2):184–192
18. Ashrafi A, Berman O (1992) Optimal design of large software systems considering reliability and cost. IEEE Trans Reliability 41(2):281–287
19. Berman O, Ashrafi A (1993) Optimization models for reliability of modular software systems. IEEE Transactions on Software Engineering 19(11):1119–1123
20. Kumar UD (1998) Reliability analysis of fault tolerant recovery blocks. OPSEARCH, J Oper Res Soc India 35(4):281–294
21. Ashrafi A, Berman O, Cutler M (1994) Optimal design of large software systems using N-version programming. IEEE Trans Reliability 43(2):344–350
22. Berman O, Kumar UD (1999) Optimization models for recovery block schemes. Eur J Oper Res 115:368–379
23. Kapur PK, Bardhan AK, Shatnawi O (2002) Why software reliability growth modeling should define errors of different severity. J Indian Stat Assoc 40(2):119–142
24. Scott RK, Gault JW, McAllister DF, Wiggs J (1984) Experimental validation of six fault-tolerant software reliability models. In: Proceedings of the IEEE 14th fault-tolerant computing symposium, pp 102–107
25. Eckhardt D, Lee L (1985) A theoretical basis for the analysis of multi-version software subject to coincident errors. IEEE Trans Softw Eng SE-11(12):1511–1517
26. Littlewood B, Miller DR (1989) Conceptual modeling of coincident failures in multi-version software. IEEE Trans Softw Eng 15(12):1596–1614
27. Dugan JB, Lyu MR (1994) System reliability analysis of an N-version programming application. IEEE Trans Reliability 43(4):513–519
28. Kanoun K, Kaaniche M, Beounes C (1993) Reliability growth of fault-tolerant software. IEEE Trans Reliability 42(2):205–218
29. Chatterjee S, Misra RB, Alam SS (2004) N-version programming with imperfect debugging. Comput Electr Eng 30:453–463
30. Kapur PK, Gupta A, Jha PC (2007) Reliability growth modeling and optimal release policy of a n-version programming system incorporating the effect of fault removal efficiency. Int J Autom Comput., Springer, Heidelberg 4(4):369–379
31. Teng X, Pham H (2002) A software reliability growth model for N-version programming systems. IEEE Trans Reliability 51(3):311–321

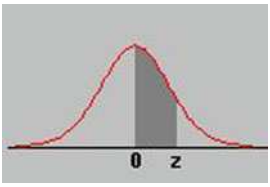
32. Zhang XM, Jeske DR, Pham H (2002) Calibrating software reliability models when the test environment does not match the user environment. Appl Stoch Models Bus Indus 18:87–99
33. Kapur PK, Kumar D, Gupta A, Jha PC (2006) On how to model software reliability growth in the presence of imperfect debugging and fault generation. In: Proceedings of the 2nd international conference on reliability and safety engineering, INCRESE, pp 261–268
34. Pham H (2006) System software reliability, Reliability Engineering Series. Springer, London
35. Kapur PK, Gupta A, Gupta D, Jha PC (2008) Optimum software release policy under fuzzy environment for a n-version programming system using a discrete software reliability growth model incorporating the effect of fault removal efficiency. Verma AK, Kapur PK, Ghadge SG (eds) Advances in performance and safety of complex system. Macmillan Advance Research Series, 803–816
36. Kapur PK, Jha PC, Bardhan AK (2002) Optimal component selection for fault tolerant COTS based software system. Presented at the international conference on operational research for development (ICORD'2002), Anna University, Chennai

Appendix A

A.1 Standard Normal (Z) Table

The Standard Normal distribution is used in various hypothesis tests including tests on single means, the difference between two means, and tests on proportions. The Standard Normal distribution has a mean of 0 and a standard deviation of 1. As shown in the illustration below, the values inside the given table represent the areas under the standard normal curve for values between 0 and the relative z-score. For example, to determine the area under the curve between 0 and 2.36, look in the intersecting cell for the row labeled 2.30 and the column labeled 0.06. The area under the curve is 0.4909. To determine the area between 0 and a negative value, look in the intersecting cell of the row and column which sums to the absolute value of the number in question. For example, the area under the curve between -1.3 and 0 is equal to the area under the curve between 0 and 1.3, so look at the cell on the 1.3 row and the 0.00 column (the area is 0.4032).

Area Between 0 and z



Z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279	0.0319	0.0359
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675	0.0714	0.0753
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064	0.1103	0.1141
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443	0.1480	0.1517
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808	0.1844	0.1879
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
0.6	0.2257	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486	0.2517	0.2549
0.7	0.2580	0.2611	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794	0.2823	0.2852
0.8	0.2881	0.2910	0.2939	0.2967	0.2995	0.3023	0.3051	0.3078	0.3106	0.3133
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340	0.3365	0.3389
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790	0.3810	0.3830
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980	0.3997	0.4015
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147	0.4162	0.4177
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292	0.4306	0.4319
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525	0.4535	0.4545
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616	0.4625	0.4633
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693	0.4699	0.4706
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4778	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4817
2.1	0.4821	0.4826	0.4830	0.4834	0.4838	0.4842	0.4846	0.4850	0.4854	0.4857
2.2	0.4861	0.4864	0.4868	0.4871	0.4875	0.4878	0.4881	0.4884	0.4887	0.4890
2.3	0.4893	0.4896	0.4898	0.4901	0.4904	0.4906	0.4909	0.4911	0.4913	0.4916
2.4	0.4918	0.4920	0.4922	0.4925	0.4927	0.4929	0.4931	0.4932	0.4934	0.4936
2.5	0.4938	0.4940	0.4941	0.4943	0.4945	0.4946	0.4948	0.4949	0.4951	0.4952
2.6	0.4953	0.4955	0.4956	0.4957	0.4959	0.4960	0.4961	0.4962	0.4963	0.4964
2.7	0.4965	0.4966	0.4967	0.4968	0.4969	0.4970	0.4971	0.4972	0.4973	0.4974
2.8	0.4974	0.4975	0.4976	0.4977	0.4977	0.4978	0.4979	0.4979	0.4980	0.4981
2.9	0.4981	0.4982	0.4982	0.4983	0.4984	0.4984	0.4985	0.4985	0.4986	0.4986
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990

A.2 Kolmogorov–Smirnov Test Table

The value in the table represents $d_{n,\alpha}$, where n is the sample size and α is the level of significance.

Sample size(n)	Level of significance A for $d_n = \sup_{-\infty<x<\infty} \lvert F_n(x) - F_0(x) \rvert$				
	0.20	0.15	0.10	0.05	0.01
1	0.900	0.925	0.950	0.975	0.995
2	0.684	0.726	0.776	0.842	0.929
3	0.565	0.597	0.642	0.708	0.828
4	0.494	0.525	0.564	0.624	0.733
5	0.446	0.474	0.510	0.565	0.669
6	0.410	0.436	0.470	0.521	0.618
7	0.381	0.405	0.438	0.486	0.577
8	0.358	0.381	0.411	0.457	0.543
9	0.339	0.360	0.388	0.432	0.514
10	0.322	0.342	0.368	0.410	0.490
11	0.307	0.326	0.352	0.391	0.468
12	0.295	0.313	0.338	0.375	0.450
13	0.284	0.302	0.325	0.361	0.433
14	0.274	0.292	0.314	0.349	0.418
15	0.266	0.283	0.304	0.338	0.404
16	0.258	0.274	0.295	0.328	0.392
17	0.250	0.266	0.286	0.318	0.381
18	0.244	0.259	0.278	0.309	0.371
19	0.237	0.252	0.272	0.301	0.363
20	0.231	0.246	0.264	0.294	0.356
25	0.210	0.220	0.240	0.270	0.320
30	0.190	0.200	0.220	0.240	0.290
35	0.180	0.190	0.210	0.230	0.270
Over 35	$\frac{1.07}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.63}{\sqrt{n}}$

Appendix B

B.1 Preliminary Concepts of Fuzzy Set Theory

Fuzzy Set

Let X be the universe whose generic element is denoted by x. A fuzzy set A in X is a function $A{:}X \rightarrow [0, 1]$. Fuzzy set A is characterized by its membership function $\mu_A{:}\mathrm{X} \rightarrow [0, 1]$ which, associates with each x in X, a real number $\mu_A(\mathrm{x})$ in [0,1] representing the grade of x in A.

Support of a Fuzzy Set

The support of a fuzzy set A in X, denoted by $S(A)$, is the crisp set given by $S(A) = \{x \in X{:}\mu_A(x) > 0\}$.

Normal Fuzzy Set

The height $h(A)$ of a fuzzy set A is defined as

$$h(A) = \sup_{x \in X} \mu_A(x) > 0 \Big\}$$

if $h(A) = 1$, then the fuzzy set A is called a normal fuzzy set, otherwise subnormal which can be normalized as

$$\frac{\mu_A(x)}{h(A)},\ x \in X.$$

Standard Union

The standard union of two fuzzy sets A and B is a fuzzy set C whose membership function is given by $\mu_C(x) = \max(\mu_A(x), \mu_B(x))$ for all $x \in X$. This we express as $C = A \cup B$.

Standard Intersection

The standard intersection of two fuzzy sets A and B is a fuzzy set D whose membership function is given by $\mu_D(x) = \min(\mu_A(x), \mu_B(x))$ for all $x \in X$. This we express as $C = A \cap B$.

α-Cut

The α-cut of the fuzzy set A in X is the crisp set A_α given by $A_\alpha = \{x \in X : \mu_A(x) > \alpha\}$ where $\alpha \in (0, 1]$.

Convex Fuzzy Set

A fuzzy set A in $\mathbb{R}^n$ is said to be a convex fuzzy set if its α-cuts A_α are (crisp) convex sets for all $\alpha \in (0, 1]$.

Theorem 1 A fuzzy set A in $\mathbb{R}^n$ is said to be a convex fuzzy iff for all $x_1, x_2 \in \mathbb{R}^n$ and $0 \leq \lambda \leq 1$

$$\mu_A(\lambda x_1 + (1 - \lambda x_2)) \geq \min(\mu_A(x_1), \mu_A(x_2))$$

Zadeh's Extension Principle

Let $f: X \to Y$ be a crisp function and F(X)(F(Y)) be the set of all fuzzy sets of X(Y). The function $f: X \to Y$ induces two functions $f: F(X) \to F(Y)$ and $f^{-1}: F(X) \to F(Y)$. The extension principle gives formulas to compute the membership function of fuzzy sets $f(A)$ in Y ($f^{-1}(B)$ in X) in terms of membership function of fuzzy set A in X (B in Y). The principle states that

1.

$$\mu_{f(A)}(y) = \sup(\mu_A(x)),\ \forall A \in F(X)$$

2.

$$\mu_{f^{-1}(B)}(x) = \mu_B(x),\ \forall B \in F(Y)$$

If the function f maps a n-tuple in X to a point in Y and $f{:}X \to Y$ given by $y = f(x_1, x_2, \ldots, x_n)$. Let $A_1, A_2, \ldots, A_n$ be n fuzzy sets in $X_1, X_2, \ldots, X_n$ respectively. The extension principle of Zadeh allows to extend the crisp function $y = f(x_1, x_2, \ldots, x_n)$ to act on n fuzzy subsets of X, namely $A_1, A_2, \ldots, A_n$ such that $B = f(A_1, A_2, \ldots, A_n)$.

The fuzzy set B is defined as

$$B = \{(y, \mu_B(y)) : y = f(x_1, x_2, \ldots, x_n), (x_1, x_2, \ldots, x_n) \in X_1 * \ldots * X_n\}$$

$$\text{and } \mu_B(y) = \sup_{x \in X, y \in f(x)} \min\left(\mu_{A_1}(x_1), \ldots, \mu_{A_n}(x_n)\right)$$

B.1.1 Fuzzy Number

A fuzzy set A in $\mathbb{R}$ is called a fuzzy number if is satisfies the following conditions

1. A is normal
2. A is convex
3. μ_A is upper semi-continuous
4. Support of A is bounded

Theorem 2 Let A be a fuzzy set in $\mathbb{R}$. Then A is a fuzzy number if and only if there exists a closed interval (which may be singleton) $[a, b] \neq \Phi$ such that

$$\mu_A(x) = \left\{ \begin{array}{ll} 1, & x \in [a, b] \\ l(x), & x \in [-\infty, a] \\ r(x), & x \in [b, \infty] \end{array} \right\}$$

where

1. $l{:}(-\infty, a) \to [0, 1]$ is non-decreasing, continuous from the right and $l(x) = 0$ for $x \in (-\infty, w_1)$, $w_1 < a$
2. $r{:}(b, \infty) \to [0, 1]$ is non-increasing, continuous from the left and $r(x) = 0$ for $x \in (w_2, \infty)$, $w_2 > b$ and $\mu_A(x)$ is called '*Membership Function*' of fuzzy set A on $\mathbb{R}$.

An element mapping to the value 0 means that the member is not included in the given set, 1 describes a fully included member. Values strictly between 0 and 1 characterize the fuzzy members. Figure B.1 illustrate a fuzzy set graphically.

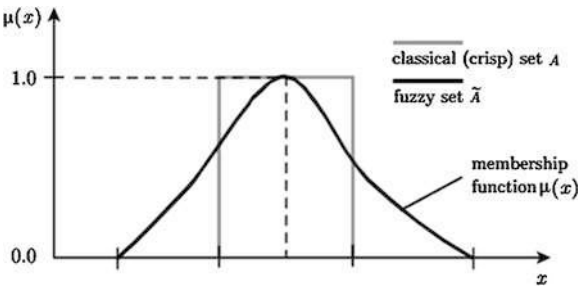


Fig. B.1 A fuzzy set

B.1.2 Triangular Fuzzy Number (TFN)

A fuzzy number A denoted by the triplet $A = (a_1, a, a_u)$ having the shape of a triangle is called a TFN. The α-cut of a TFN is the closed interval $A_\alpha = [a_\alpha^L, a_\alpha^R] = [(a - a_1)\alpha + a_1, (a - a_u)\alpha + a_u]$, $\alpha \in (0, 1]$ and its membership function μ_A is given by

$$\mu_A(x) = \left\{ \begin{array}{ll} 0, & x < a_l, x > a_u, \\ (x - a_l)/(a - a_l), & a_l \leq x \leq a, \\ (a_u - x)/(a_u - a), & a < x \leq a_u \end{array} \right\}$$

B.1.3 Ranking of Fuzzy Numbers

Ranking of fuzzy number is an important issue in the study of fuzzy set theory and is useful in various applications. Fuzzy mathematical programming is one of the applications. There are numerous methods proposed in literature for ranking the fuzzy numbers such as ranking function (index) approach, k-preference index approach and possibility theory approach, useful in particular context but not in general. We use the Ranking function (index) approach for ranking the fuzzy numbers for our problem.

B.1.3.1 Ranking Function (Index) Approach

Let $\mathrm{N}(\mathbb{R})$ be the set of all fuzzy numbers in $\mathbb{R}$ and $A, B, \in N(\mathbb{R})$. Define a function $F : N(\mathbb{R}) \rightarrow \mathbb{R}$, called a ranking function or ranking index, where $F(A) \leq F(B)$ is equivalent to A $(\leq)$ B. Following indices are proposed by Yager ().

1. $F_1(A) = \left(\int_{a_1}^{a_u} x\mu_{A(x)dx}\right) \Big/ \left(\int_{a_1}^{a_u} \mu_{A(x)dx}\right)$, Where a1 and au are the lower and upper limits of the support of A. The value $F_1(A)$ is the centroid of the fuzzy number $A \in N(\mathbb{R})$. For example, If $A = (a_1, a, a_u)$ is a triangular fuzzy number (TFN) where a_1 and a_u are the lower and upper limits of the support of A and a is the model value then $F_1(A) = (a_1 + a + a_u)/3$.
2. $F_2(A) = \left(\int_0^{a_{\max}} m\left[a_\alpha^L, a_\alpha^R\right] d\alpha\right)$, Where $\alpha_{\max}$ is the height of A, $A_\alpha = [a_\alpha^L, a_\alpha^R]$ is a α-cut, $\alpha \in (0, 1]$, and $m[a_\alpha^L, a_\alpha^R]$ is the mean value of elements of the α-cut. For example, If $A = (a_1, a, a_u)$ is a TFN, $\alpha_{\max} = 1$ and $A_\alpha = [a_\alpha^L, a_\alpha^R] = [(a - a_1)\alpha + a_1, (a - a_u)\alpha + a_u]$ then $m[a_\alpha^L, a_\alpha^R] = ((2a - a_1 - a_u)\alpha + (a_1 + a_u))/((2a - a_1 - a_u)\alpha + (a_1 + a_u))/2$ and $F_2(A) = (a_1 + 2a + a_u)/4$.

Appendix C

C.1 Mean Value Functions of Failure and Removal Phenomenon for Faults of Type AC, BC, B and C

C.1.1 Continuous Time SRGM for the 3VP System

Mean value functions of the removal phenomena of fault type *AC, BC, B* and *A* are

$$m_{AC,r}(t) = a_{AC}\left(1 - e^{-bp_{13}t}\right) + \frac{\bar{p}_{13}a_{ABC}}{p_{13}\bar{p}_2}\left(p_2 e^{-bp_{13}t} - e^{-bp_{123}t} + \bar{p}_2\right)$$

$$m_{BC,r}(t) = a_{BC}\left(1 - e^{-bp_{23}t}\right) + \frac{\bar{p}_{23}a_{ABC}}{p_{23}\bar{p}_1}\left(p_1 e^{-bp_{23}t} - e^{-bp_{123}t} + \bar{p}_1\right)$$

$$\begin{aligned} m_{B,r}(t) = {} & \left(\frac{1}{1-\alpha_2}\right)\left\{a_B + (1 - p_2(1-\alpha_2))\left[\frac{a_{ABC}}{p_2} + \sum_{(i,V)=(1,A),(3,C)}\left(\frac{a_{BV}}{p_2} + \frac{\bar{p}_{2i}}{p_2^2 p_i}a_{ABC}\right)\right]\right\} \\ & \times \left(1 - e^{-b(1-\alpha_2)p_2 t}\right) \\ & + \left\{\frac{a_{ABC}(1 - p_2(1-\alpha_2))}{(p_{13} - (1-\alpha_2))}\right\}\left\{\frac{1}{p_2} + \frac{\bar{p}_{21}}{p_2^2 p_1 \bar{p}_3} + \frac{\bar{p}_{23}}{p_2^2 p_3 \bar{p}_2}\right\}\left(e^{-bp_{123}t} - e^{-b(1-\alpha_2)p_2 t}\right) \\ & + \left(\frac{(1 - p_2(1-\alpha_2))}{(p_1 - (1-\alpha_2))}\right)\left\{\left(\frac{a_{AB}}{p_2} + \frac{\bar{p}_{21}p_3}{p_2^2 p_1 \bar{p}_3}a_{ABC}\right)\left(e^{-bp_{21}t} - e^{-b(1-\alpha_2)p_2 t}\right)\right\} \\ & + \left(\frac{(1 - p_2(1-\alpha_2))}{(p_3 - (1-\alpha_1))}\right)\left\{\left(\frac{a_{BC}}{p_2} + \frac{\bar{p}_{23}p_1}{p_2^2 p_3 \bar{p}_1}a_{ABC}\right)\left(e^{-bp_{23}t} - e^{-b(1-\alpha_2)p_2 t}\right)\right\} \end{aligned}$$

$$
\begin{aligned}
m_{C,r}(n) = \left(\frac{1}{1-\alpha_3}\right)\Bigg\{ & a_C + (1-p_3(1-\alpha_3)\left[\frac{a_{ABC}}{p_3} + \sum_{(i,V)=(1,A),(2,B)}\left(\frac{a_{CV}}{p_3} + \frac{\bar{p}_{3i}}{p_3^2 p_i}a_{ABC}\right)\right]\Bigg\} \\
& \times\left(1-e^{-b(1-\alpha_3)p_3 t}\right) \\
& + \left\{\frac{a_{ABC}(1-p_3(1-\alpha_3))}{(p_{12}-(1-\alpha_3))}\right\}\left\{\frac{1}{p_3} + \frac{\bar{p}_{31}}{p_3^2 p_1 \bar{p}_2} + \frac{\bar{p}_{32}}{p_3^2 p_2 \bar{p}_1}\right\}\left(e^{-bp_{123}t} - e^{-b(1-\alpha_3)p_3 t}\right) \\
& + \left(\frac{(1-p_3(1-\alpha_3))}{(p_2-(1-\alpha_1))}\right)\left\{\left(\frac{a_{BC}}{p_3} + \frac{\bar{p}_{32}p_1}{p_3^2 p_2 \bar{p}_1}a_{ABC}\right)\left(e^{-bp_{32}t} - e^{-b(1-\alpha_3)p_3 t}\right)\right\} \\
& + \left(\frac{(1-p_3(1-\alpha_3))}{(p_1-(1-\alpha_1))}\right)\left\{\left(\frac{a_{AC}}{p_3} + \frac{\bar{p}_{13}p_2}{p_3^2 p_1 \bar{p}_2}a_{ABC}\right)\left(e^{-bp_{13}t} - e^{-b(1-\alpha_3)p_3 t}\right)\right\}
\end{aligned}
$$

Note here $a_{ij} = a_{ji}$ and $p_{ij} = p_{ji}$.

The mean value functions of the failure phenomena can be obtained from the corresponding mean value function of the removal phenomena using the relation $m_r(t) = pm_f(t)$.

C.1.2 Discrete Time SRGM for the 3VP System

Mean value functions of the removal phenomena of fault type *AC, BC, B* and *A* are

$$
\begin{aligned}
m_{AC,r}(n) &= a_{AC}(1-(1-bp_{13}\delta)^n) \\
&\quad + \frac{a_{ABC}\bar{p}_{13}}{p_{13}\bar{p}_2}(p_2(1-bp_{13}\delta)^n - (1-bp_{123}\delta)^n + \bar{p}_2) \\
m_{BC,r}(n) &= a_{BC}(1-(1-bp_{23}\delta)^n) \\
&\quad + \frac{a_{ABC}\bar{p}_{23}}{p_{23}\bar{p}_1}(p_1(1-bp_{23}\delta)^n - (1-bp_{123}\delta)^n + \bar{p}_1)
\end{aligned}
$$

$$m_{B,r}(n) = \left(\frac{1}{1-\alpha_2}\right)\left\{a_B + (1-p_2(1-\alpha_2)\left[\frac{a_{ABC}}{p_2} + \sum_{(i,V)=(1,A),(3,C)}\left(\frac{a_{BV}}{p_2} + \frac{\bar{p}_{2i}}{p_2^2 p_i}a_{ABC}\right)\right]\right\}$$
$$\times (1-(1-bp_2\delta(1-\alpha_2))^n)$$
$$+\left\{\frac{a_{ABC}(1-p_2(1-\alpha_2))}{(p_{13}-(1-\alpha_2))}\right\}\left\{\frac{1}{p_2} + \frac{\bar{p}_{21}}{p_2^2 p_1 \bar{p}_3} + \frac{\bar{p}_{23}}{p_2^2 p_3 \bar{p}_2}\right\}((1-bp_{123}\delta)^n - (1-bp_2\delta(1-\alpha_2))^n)$$
$$+\left(\frac{(1-p_2(1-\alpha_2))}{(p_1-(1-\alpha_2))}\right)\left\{\left(\frac{a_{AB}}{p_2} + \frac{\bar{p}_{21}p_3}{p_2^2 p_1 \bar{p}_3}a_{ABC}\right)((1-bp_{21}\delta)^n - (1-bp_2\delta(1-\alpha_2))^n)\right\}$$
$$+\left(\frac{(1-p_2(1-\alpha_2))}{(p_3-(1-\alpha_1))}\right)\left\{\left(\frac{a_{BC}}{p_2} + \frac{\bar{p}_{23}p_1}{p_2^2 p_3 \bar{p}_1}a_{ABC}\right)((1-bp_{23}\delta)^n - (1-bp_2\delta(1-\alpha_2))^n)\right\}$$

$$m_{C,r}(n) = \left(\frac{1}{1-\alpha_3}\right)$$
$$\left\{a_C + (1-p_3(1-\alpha_3))\left[\frac{a_{ABC}}{p_3} + \sum_{(i,V)=(1,A),(2,B)}\left(\frac{a_{CV}}{p_3} + \frac{\bar{p}_{3i}}{p_3^2 p_i}a_{ABC}\right)\right]\right\}(1-(1-bp_3\delta(1-\alpha_3))^n)$$
$$+\left\{\frac{a_{ABC}(1-p_3(1-\alpha_3))}{(p_{12}-(1-\alpha_3))}\right\}\left\{\frac{1}{p_3} + \frac{\bar{p}_{31}}{p_3^2 p_1 \bar{p}_2} + \frac{\bar{p}_{32}}{p_3^2 p_2 \bar{p}_1}\right\}$$
$$((1-bp_{123}\delta)^n - (1-bp_2\delta(1-\alpha_2))^n)$$
$$+\left(\frac{(1-p_3(1-\alpha_3))}{(p_2-(1-\alpha_1))}\right)\left\{\left(\frac{a_{BC}}{p_3} + \frac{\bar{p}_{32}p_1}{p_3^2 p_2 \bar{p}_1}a_{ABC}\right)((1-bp_{32}\delta)^n - (1-bp_3\delta(1-\alpha_3))^n)\right\}$$
$$+\left(\frac{(1-p_3(1-\alpha_3))}{(p_1-(1-\alpha_1))}\right)\left\{\left(\frac{a_{AC}}{p_3} + \frac{\bar{p}_{13}p_2}{p_3^2 p_1 \bar{p}_2}a_{ABC}\right)((1-bp_{13}\delta)^n - (1-bp_3\delta(1-\alpha_3))^n)\right\}$$

Here again $a_{ij} = a_{ji}$ and $p_{ij} = p_{ji}$ and the mean value functions of the failure phenomena can be obtained from the corresponding mean value function of the removal phenomena using the relation $m_r(n) = pm_f(n)$.

Answers to the Selected Exercises

Chapter 1

1. Software Reliability Engineering (SRE) is a scientific discipline that creates and utilizes sound engineering principles in order to economically obtain software systems that are not only reliable but also work proficiently on real machines. The IEEE society has defined SRE as widely applicable, standard and proven practice that apply systematic, disciplined, quantifiable approach to the software development, test, operation, maintenance and evolution with emphasis on reliability and the study in these approaches. The software engineering is concerned with scheduling and systematizing the software development process to monitor the progress of the various stages of software development using its tools, methods and process to engineer quality software and maintaining a tight control throughout the development process. SRE broadly focuses on quantitatively characterizing the following standardized six quality characteristics defined by ISO/IEC: functionality, usability, reliability, efficiency, maintainability and portability. One of the major roles of SRE lies in assuring and measuring the reliability of the software.
   SRE management techniques work by applying two fundamental philosophies
   - Deliver the desired functionality more efficiently by quantitatively characterizing the expected use, use this information to optimize the resource usage focusing on the most used and/or critical functions and make testing environment representative of operational environment.
   - Balances customer needs for reliability, time and cost effectiveness. It works by setting quantitative reliability, schedule and cost objectives and engineers' strategies to meet these objectives.
2. Refer to figure 1.1 and Pressman (2005)

3. The main limitations for the waterfall model includes
   - The model implies that you should attempt to complete a given stage before moving on to the next stage
   - Does not account for the fact that requirements constantly change.
   - Freezing the requirements usually requires choosing the hardware (since it forms a part of the requirement specification). A large project might take a few years to complete. If the hardware is selected early, then due to the speed at which hardware technology is changing, it is quite likely that the final software will employ a hardware technology that is on the verge of becoming obsolete. This is clearly not desirable for such expensive software.
   - It also means that customers can not use anything until the entire system is complete.
   - The model makes no allowances for prototyping.
   - It implies that you can get the requirements right by simply writing them down and reviewing them.
   - The model implies that once the product is finished, everything else is maintenance.

   Alternative to the waterfall model one can choose Prototyping Software Life Cycle Model or Iterative Enhancement Life Cycle Model

5. Software Reliability is the accepted as key characteristic of software quality since it quantifies software failures – the most unwanted events and hence is of major concern to the software developers as well as users. Further it is the multidimensional property including other customer satisfaction factors such as functionality, usability, performance, serviceability, capability, installability, maintainability and documentation. For this reason it is considered to be a "must be quality" of the software. Other measures of software quality are software availability, software maintainability, mean time to failure, mean time between failures etc.
6. Statistical testing aims to measure software reliability rather than discovering faults. It is an effective sampling method to assess system reliability and hence also known as reliability testing. Data collected from other test methods is used here to predict the reliability level achieved and which can further be used to depict the time when the desired level of quality in terms of reliability can be achieved. Reliability assessment is of undue importance to both the developers and user; it provides a quantitative measure of the number of remaining faults, failure intensity, and a number of decisions related to the development, release and maintenance of the software in the operational phase. To the users it provides a measure for having confidence in the software quality and their level of acceptability.
7. Refer to section 1.5.1
9. Refer to figure 1.4
10. Refer to section 1.3

13. The reliability function is $R(t) = \int_t^{\infty} \frac{1}{\sigma\sqrt{2\prod}} e^{-\frac{1}{2}\left(\frac{s-\mu}{\sigma}\right)^2} ds$
14. Refer section 1.5.5
15. Refer figure 1.12
16. Refer section 1.6.1
17. Refer section 1.6.1
18. See reference Abramson M A, Chrissis J W (1998) Sequential quadratic programming and the ASTROS structural optimization system. Structural Optimization 15:24–32
19. Refer section 1.7.3

Chapter 2

2. Exponential curve describes a uniform operational profile whereas an S-shaped curve describes a non uniform operational profile
3. a. $m_r(t) = a\left(1 - (1 + bt)e^{-bt}\right)$

   b. $m_r(t) = a\left[\frac{1-e^{-bt}}{1+\beta e^{-bt}}\right]$
4. Refer to section 2.7
5. Refer to section 2.7.2
6. The test effort based models are

$$m(t) = a\left(1 - e^{-bW(t)}\right)$$

$$m_r(t) = a\left(1 - (1 + bW(t))e^{-bW(t)}\right)$$

$$m_r(t) = a\left[\frac{1 - e^{-(p+q)W(t)}}{1 + (q/p)e^{-(p+q)W(t)}}\right]$$

7. The data set used in chapter 8 is used to estimate the parameters of the models analyzed in section 2.9.1. The estimated values of the parameters are given in the following table.

Model	Estimated Parameters						Comparison Criteria	
							MSE	R^2
M1	5611 (a)	0.0010 (λ_0)	-	-	-	-	17.82	0.995
M2	1753 (a)	0.0033 (b)	-	-	-	-	18.66	0.994
M3	1610 (a)	0.0023 (b_1)	0.0040 (b_2)	0.2192 (p_1)	0.7808 (p_1)	-	20.98	0.994
M4	225 (a)	0.0881 (b)	-	-	-	-	20.75	0.994
M5	229 (a)	0.0187 (u_i)	0.0876 (u_f)	-	-	-	6.38	0.998
M6	229 (a)	0.0187 (p)	0.0689 (q)	-	-	-	6.38	0.998
M7	334 (a) 0.2872 (p_1)	0.0361 (b_1) 0.5614 (p_2)	0.0621 (b_2) 0.1513 (p_3)	0.1329 (b_3)	-	-	8.16	0.998
M8	216 (a) 0.1924 (p_1)	0.1280 (b_1) 0.0774 (p_2)	0.5709 (b_2) 0.7301 (p_3)	0.1330 (b_3)	83.1192 (β_1)	4.3708 (β_2)	7.68	0.998
M9	17 (a)	0.0601 (b)	0.0477 (c)	-	-	-	7.58	0.998
M10	1211 (a)	0.9591 (b)	0.1879 (c)	0.0283 (p_1)	0.9717 (p_2)	-	29.08	0.992

Chapter 3

2. If the fault is removed perfectly with no new fault generation then the fault content will decrease by one, however if the fault has been repaired imperfectly with no new fault introduction then the fault content will remain same as earlier, while if the current fault is removed imperfectly and some new fault is also manifested then the fault content will increase.
3. The mean value function of the SRGM is

$$m(t) = \left[(a+k)\left(1-e^{-bt}\right) - \frac{kb}{b-\theta}\left(e^{-\theta t} - e^{-bt}\right)\right]$$

4. See section 3.5.2

6. $m_r(t) = \frac{a}{1-\alpha}\left[1 - e^{-bp(1-\alpha)W(t)}\right]$
7. Refer section 3.7
8. The mean value function of the SRGM with the usage function is $W(t) = r + st^k$ is $m_f(t) = \frac{a}{p(1-\alpha)}\left[1 - \left(\frac{(1+\beta)e^{-b\left(r+st^k\right)}}{1+\beta e^{-b\left(r+st^k\right)}}\right)^{p(1-\alpha)}\right]$

The estimated values of the parameters are given in the following table

Estimated Parameters								Comparison Criteria	
a	b	β	p	α	r	s	k	MSE	R^2
36	0.2578	219.99	0.9987	0.0012	5.04	0.5837	0.7788	2.32	0.993

Chapter 4

1. Refer to introduction section.
2. Refer to introduction section.
3. Refer to introduction section.
4. Both coverage functions generate an s-shaped curve. The later converges slowly as compared to the former. This type of curve gives better result if the test strategy is less effective in attaining maximum coverage.
6. The mean value function of the SRGM is

$$m(t) = a\left(\frac{1 - e^{-\delta\frac{(X(t))^{k+1}}{k+1}}}{1 + \beta e^{-\delta\frac{(X(t))^{k+1}}{k+1}}}\right); \quad \delta = b_3(b_1 + b_2) \text{ and } \beta = b_2/b_1$$

7. The software reliability growth model based on the model discussed in section 4.3.2 applicable to this software will be

$$m_r(t) = a_1(1 - e^{-b_1 t}) + \frac{a_2}{1 + \beta_2 e^{-b_2 t}}\left(1 + \frac{b_2 e^{-v_2 t} - v_2 e^{-b_2 t}}{v_2 - b_2}\right) + \frac{a_3}{1 + \beta_3 e^{-b_3 t}}\left(\begin{array}{l} 1 + \frac{b_3}{v_3 - b_3}\left(v_3 t + \frac{2v_3 - b_3}{v_3 - b_3}\right)e^{-v_3 t} \\ -\left(1 + \frac{b_3(2v_3 - b_3)}{(v_3 - b_3)^2}\right)e^{-b_3 t} \end{array}\right);$$

$$v_2 \neq b_2, \quad v_3 \neq b_3,$$

8. Estimated values of the parameters are

Estimated Parameters										Comparison Criteria	
a_1	a_2	a_3	b_1	b_2	b_3	β_2	β_3	v_1	v_2	MSE	Variation
27	57	56	0.8272	0.8774	0.6507	80.24	86.92	0.6374	0.6221	138.41	12.55

Chapter 5

1. Refer to Introduction section.
2. Refer to Introduction section.
3. Refer to Introduction section.
4. The mean value functions for the SRGM is given as

$$m_r(t) = \begin{cases} \dfrac{a}{(1-\alpha_1)}\left[1 - \left(\dfrac{(1+\beta)\exp(-b_1 pt)}{1+\beta\exp(-b_1 pt)}\right)^{(1-\alpha_1)}\right] & 0 \le t \le \tau, \\ \dfrac{a}{(1-\alpha_2)}\left[1 - \left(\left(\dfrac{1+\beta e^{-b_1\tau p}}{1+\beta}\right)^{-(1-\alpha_1)}\left(\dfrac{1+\beta e^{-b_2 tp}}{1+\beta e^{-b_2\tau p}}\right)^{-(1-\alpha_2)}\right.\right. & \\ \left.\left. e^{-b_1\tau p(1-\alpha_1)-b_2(t-\tau)p(1-\alpha_2)}\right)\right] & \\ + \dfrac{a(\alpha_1-\alpha_2)}{(1-\alpha_1)(1-\alpha_2)}\left[1 - \left(\dfrac{(1+\beta)e^{-b_1\tau p}}{1+\beta e^{-b_1\tau p}}\right)^{(1-\alpha_1)}\right] & t > \tau \end{cases}$$

6. The differential equation of the model is

$$\frac{m'(t)}{w(t)} = b(t)[a - m(t)]$$

where $$b(t) = \begin{cases} \dfrac{b_1^2 W(t)}{1 + b_1 W(t)} & 0 \le t \le \tau \\ \dfrac{b_2^2 W(t)}{1 + b_2 W(t)} & t > \tau \end{cases}$$

and the mean value function is

$$m(t) = \begin{cases} a\left(1 - (1 + b_1 W(t))e^{-b_1 W(t)}\right) & 0 \le t \le \tau, \\ m(t) = a\left[1 - \left(\dfrac{1 + b_1 W(\tau)}{1 + b_2 W(\tau)}\right)\begin{pmatrix}(1 + b_2 W(t)) \\ e^{-(b_1 W(\tau) + b_2 W(t-\tau))}\end{pmatrix}\right] & t > \tau \end{cases}$$

where $W(t - \tau) = W(t) - W(\tau)$

7. The Logistic test effort function fits best on this data as shown in application 5.11.1. Using the data analysis results of the logistic testing effort function the unknown estimates of the model developed in exercise 6 are given in the following table.

Estimated Parameters			Comparison Criteria	
a_1	b_1	b_2	MSE	R^2
386	0.0902	0.0713	149.40	0.995

This model gives better estimation result on this data set as compared to the exponential test effort based change point model.

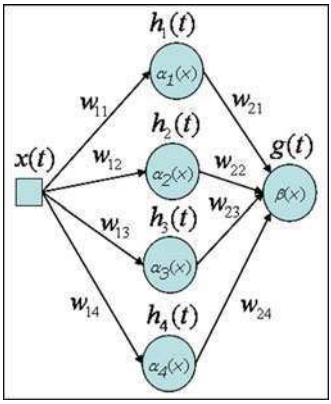
Chapter 6

1. Refer to the introduction section.
2. Mean value function for the isolation and removal process is given as

$$m(t) = a\left(\left(-e^{\left(-bt + \mu b + (b\sigma)^2/2\right)}\Phi\left((0,t), b\sigma^2 + \mu, \sigma\right)\right) + \Phi\left((0,t), \mu, \sigma\right)\right)$$

Chapter 7

1. Refer to section 7.2.
2. See paragraph, Network architecture section 7.2.
3. See paragraph, learning algorithm section 7.2.
4. Refer to section 7.3.1
5. Neural network to describe the failure process of software containing four types of faults is given as



Use the above network, the activation function for the j^{th} node in the hidden layer

$$\alpha_j(x) = 1 - e^{-x} \sum_{i=0}^{j-1} \frac{(x)^i}{i!} \quad j = 1, 2, 3, 4$$

and the activation function for the neuron in output layer $\beta(x) = x$. Now following the equation (7.4.20) to (7.4.24) we can describe the failure process.

6. The estimates of the model parameters are

Estimated Parameters								Comparison Criteria	
a_1	a_2	a_3	a_4	b_1	b_2	b_3	b_4	MSE	R^2
72	30	154	23	0.22	0.09	0.47	0.23	18.54	0.996

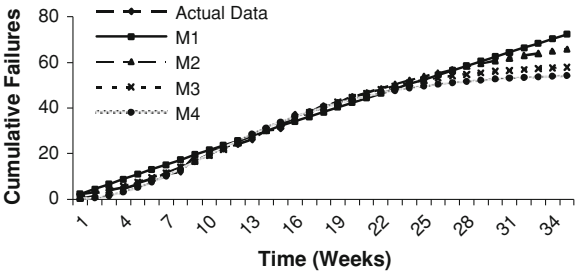
Chapter 8

1. Refer Introduction section.
2. See section 8.2
3. See section 8.3.
4. See section 8.3.1 and 8.3.2.

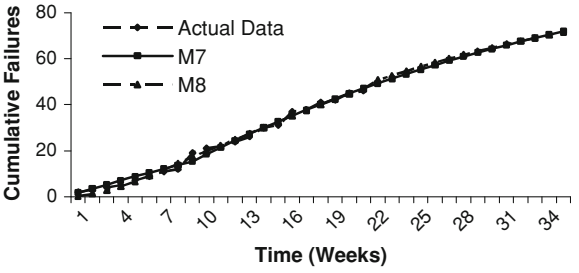
5.

Model	Estimated Parameters				Comparison Criteria		
	a	b, b_1	b_2, β	σ	MSE	R^2	RMSPE
M1	661	0.0275	-	0.2199	7.80	0.969	2.643632
M2	77	0.0966	-	1E-06	1.73	0.993	1.245796
M3	59	0.1684	8.28	1E-06	1.48	0.994	1.120364
M4	55	0.2100	-	1E-06	3.37	0.987	1.736672
M7	118	0.0154	0.0309	1E-06	2.70	0.99	1.514483
M8	91	0.0880	0.0795	1E-06	1.76	0.993	1.222912

Goodness of fit curve models M1–M4



Goodness of fit curve for change point based SDE models (M7 and M8)



Chapter 9

1. Refer section 1.5.5
2. See introduction section.
3. See introduction section.
4. See introduction section 9.2.1.

5. Single change point Discrete exponential SRGM is

$$m(n) = \begin{cases} a(1-(1-b_1\delta)^n) & 0 \le n < \eta_1 \\ a(1-(1-b_1\delta)^{n_1}(1-b_2\delta)^{n-n_1}) & n \ge \eta_1 \end{cases}$$

Estimation results for discrete exponential, Delayed S –Shaped and flexible change point SRGM

Model	Estimated Parameters				Comparison Criteria	
	a	b_1,	b_2	β	MSE	R^2
Exponential	3206	0.01084	0.030591	-	2477.23	0.982
S-shaped	1395	0.0751	0.1592	-	6047.69	0.958
Flexible	3314	0.0104	0.0292	0.00001	2567.12	0.983

Exponential and flexible models provide similar results, however the results of flexible model shows the data shows an exponential trend due to negligible value of the shape parameter $\beta = 0.00001$

6. Mean value functions for the removal process for each type of faults are

Model for Simple Faults

$$m_1(n) = \begin{cases} a_1(1-(1-b_{11})^n) & 0 \le n \le \eta_1 \\ a_1\left(1-(1-b_{11})^{\eta_1}(1-b_{12})^{(n-\eta_1)}\right) & \eta_1 < n \le \eta_2 \\ a_1\left(1-(1-b_{11})^{\eta_1}(1-b_{12})^{(\eta_2-\eta_1)}(1-b_{13})^{(n-\eta_2)}\right) & \eta_2 < n \le \eta_3 \\ a_1\left[1-\begin{pmatrix}(1-b_{11})^{\eta_1}(1-b_{12})^{(\eta_2-\eta_1)} \\ (1-b_{13})^{(\eta_3-\eta_2)}(1-b_{14})^{(n-\eta_3)}\end{pmatrix}\right] & n > \eta_3 \end{cases}$$

$$a_1 = ap_1$$

Model for Hard Faults

$$m_2(n)=\begin{cases} a_2(1-(1-b_{21})^n) & 0\le n\le \eta_1 \\ a_2\left[1-\frac{(1+b_{22}n)}{(1+b_{22}\tau_1)}(1-b_{21})^{\tau_1}(1-b_{22})^{(n-\tau_1)}\right] & \eta_1<n\le\eta_2 \\ a_2\left[1-\frac{(1+b_{22}\tau_2)}{(1+b_{22}\tau_1)}(1-b_{21})^{\tau_1}(1-b_{22})^{(\tau_2-\tau_1)}(1-b_{23})^{(n-\tau_2)}\right] & \eta_2<n\le\eta_3 \\ a_2\left[1-\frac{(1+b_{22}\tau_2)}{(1+b_{22}\tau_1)}\begin{pmatrix}(1-b_{21})^{\tau_1}(1-b_{22})^{(\tau_2-\tau_1)} \\ (1-b_{23})^{(\tau_3-\tau_2)}(1-b_{24})^{(n-\tau_3)}\end{pmatrix}\right] & n>\eta_3 \\ & a_2=ap_2 \end{cases}$$

Model for Complex Faults

$$m_2(n)=\begin{cases} a_3(1-(1-b_{31})^n) & 0\le n\le \eta_1 \\ a_3\left[1-S(1-b_{31})^{\eta_1}(1-b_{32})^{(n-\eta_1)}\right] & \eta_1<n\le\eta_2 \\ a_3\left[1-S_1\left(\frac{1+b_{33}n}{1+b_{33}\eta_2}\right)\begin{pmatrix}(1-b_{31})^{\eta_1}(1-b_{32})^{(\eta_2-\eta_1)} \\ (1-b_{33})^{(n-\eta_2)}\end{pmatrix}\right] & \eta_2<n\le\eta_3 \\ a_3\left[1-S_1\left(\frac{1+b_{33}\eta_3}{1+b_{33}\eta_2}\right)\begin{pmatrix}(1-b_{31})^{\eta_1}(1-b_{32})^{(\eta_2-\eta_1)} \\ (1-b_{33})^{(\eta_3-\eta_2)}(1-b_{34})^{(n-\eta_3)}\end{pmatrix}\right] & n>\eta_3 \end{cases}$$

$$S=\left(\frac{1+b_{32}n+\frac{b_{32}^2n(n+1)}{2}}{1+b_{32}\eta_1+\frac{b_{32}^2\eta_1(\eta_1+1)}{2}}\right),\ S_1=\left(\frac{1+b_{32}\eta_2+\frac{b_{32}^2\eta_2(\eta_2+1)}{2}}{1+b_{32}\eta_1+\frac{b_{32}^2\eta_1(\eta_1+1)}{2}}\right) \qquad a_3=ap_3$$

Chapter 10

1. See introduction section
2. See introduction section.
3. See introduction section.
4. The operational performance of a software system is to a large extent dependent on the time spent in testing. In general, the longer the testing phase, better the performance. Also, the cost of fixing a fault during testing phase is generally much lesser than during operational phase. However, the time spent in testing, delays the release of the system for operational use, and incurs additional cost. This suggests a reduction in test time and an early release of the system. The first and third component of the cost function represents the cost of fault removal and testing during the testing phase and the second component is the cost of fault removal in the operational use. If the software is tested for a longer time the value of first and third component will increase and the third component will decrease. However if an early release of the software is decided the second component will exceed over the other two. Thus considering the two conflicting objectives of better performance with longer testing and reduces costs with early release the cost model determines the optimal cost.
5. The cost function is

$$C(T) = C_1 m(T) + C_2(m(T_l) - m(T)) + C_3 T + \int_0^T p_c(T_s - t)dG(t)$$

Optimal release policies minimizing the cost function subject to reliability requirement for an exponential SRGM $m(t) = a\left(1 - e^{-bt}\right)$is given as follows. *Ts* (the scheduled delivery time) is defined a random variable with cdf *G(t)* and finite pdf *g(t)*.

Case(i) When T_s is deterministic, let $G(t) = \begin{cases} 1 \ , \ t \geq T_s \\ 0 \ , \ t < T_s \end{cases}$

Equating the first derivative of cost function to be zero

$$Q(T) \equiv (C_2 - C_1)m'(T) - \frac{d}{dT}p_0(T - T_s) = C_3$$

$Q(T_s)$ is a decreasing function in $T(T_s \leq T < \infty)$where $Q(T_s) = (C_2 - C_1)$ $m'(T_s) > 0$ *and* $Q(\infty) < 0$. Combining cost and reliability requirements and assuming $T_l > T_0$ and $T_l > T_1$ and $T_l > T_s$ the release policy is stated as
1. *if* $Q(T_s) \leq C_3$ and $R(x|T_s) \geq R_0,$ then $T* = T_s$
2. *if* $Q(T_s) \leq C_3$ and $R(x|T_s) < R_0,$ then $T* = T_1$

3. *if* $Q(T_s) > C_3$ and $R(x|T_s) \geq R_0$, then $T* = T_0$
4. if $Q(T_s) > C_3$ and $R(x|T_s) < R_0$, then $T* = \max(T_0, T_1)$

Case(ii) When T_s has an arbitrary distribution *G(t)* with finite mean μ, then equating the first derivative of cost function to be zero

$$P(T) \equiv (C_2 - C_1)m'(T) - \int_0^T \frac{d}{dT} p_c(T-t)dG(t) = C_3$$

P(T) is a decreasing function in T with $P(0) = (C_2 - C_1)m\prime(0) > 0$ *and* $P(\infty) < 0$. Combining cost and reliability requirement and assuming $T_l > T_0$ and $T_l > T_1$ the release policy is stated as

1. *if* $P(0) \leq C_3$ and $R(x|0) \geq R_0$, then $T* = 0$
2. *if* $P(0) \leq C_3$ and $R(x|0) < R_0$, then $T* = T_1$
3. if $P(0) > C_3$ and $R(x|0) \geq R_0$, then $T* = T_0$
4. *if* $P(0) > C_3$ and $R(x|0) < R_0$, then $T* = \max(T_0, T_1)$.

6. Refer to the reference Pham (1996)
8. Refer to

Jha PC, Gupta D, Gupta A, Kapur PK (2008), Release time decision policy of software employed for the safety of critical system under uncertainty. OPSEARCH, *Journal of Operational research Society of India*, 45(3):209–224.

Chapter 11

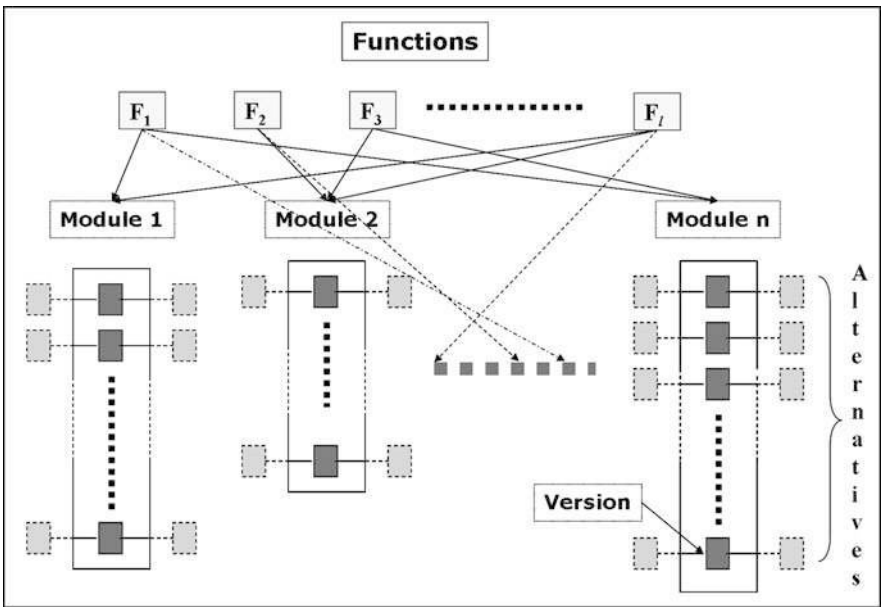
1. Refer to the introduction section
2. The resource allocation will change as given in following table.

Module	a_i	v_i	b_i (in10^{-4})	W_i^*	z_i
1	89	0.12	4.1823	6688.85	5.4
2	25	0.08	5.0923	2590.42	6.7
3	27	0.08	3.9611	2890.29	8.6
4	45	0.13	2.2956	6951.04	9.1
5	39	0.11	2.5336	5463.26	9.8
6	39	0.08	1.7246	3949.16	19.7
7	59	0.08	0.8819	4812.07	38.6
8	68	0.14	0.7274	12831.46	26.7
9	37	0.14	0.6824	3823.42	28.5
10	14	0.04	1.5309	0.00	14.0

4. Refer to Huang CY, Lo J H, Kuo S Y, Lyu M R (2004) Optimal allocation of testing-resource considering cost, reliability, and testing-effort. Proceedings 10th IEEE/IFIP Pacific Rim International Symposium on Dependable Computing, Papeete, Tahiti, French Polynesia, 103–112.
5. Refer to Jha P C, Gupta D, Yang B, Kapur P K (2009) Optimal testing-resource allocation during module testing considering cost, testing effort and reliability. Computers & Industrial Engineering 57(3):1122–1130.

Chapter 12

1. These techniques are namely- Fault avoidance/prevention during development, fault removal by means of testing and fault tolerance during use. For detail refer to introduction section.
2. Refer to introduction section 12.1 and reference Pham H (2006) System software reliability. Reliability Engineering Series, Springer Verlag, London
3. Refer to section 12.2
4. Two important design diversity techniques are—Recovery Block and N-Version Programming. For details refer to 12.2
5. Refer to the advanced design diversity techniques in section 12.2
6. See figure 12.5 and 12.6 in section 12.3
7. See section 12.4
8. The following figure portraits the component selection problem of a software that performs l different function, consisting of n modules. For performing each function different set of modules are required. For each modules different version are available. Versions differ in their design but are capable of performing same task. For each version different alternatives are available with different cost and reliability. The problem is to find the optimal component for each of the module so that either the reliability or cost or both can be optimized for the total software.



9. The problem can solved with the model without redundancy in section 12.4.1. The solution is $X_{11} = X_{23} = X_{33} = 1$ and $X_{12} = X_{13} = X_{14} = X_{21} = X_{22} = X_{31} = X_{32} = 0$. System reliability 0.897 and budget consumed = \$34.
10. If budget is \$34 the solution is same as in exercise 9 and if budget is \$50 the solution changes to $X_{11} = X_{13} = X_{21} = X_{33} = 1$ and $X_{12} = X_{14} = X_{22} = X_{23} = X_{31} = X_{32} = 0$. System reliability 0.899 and budget consumed = \$48.

Index