THE UNIVERSITY OF THE SOUTH PACIFIC
LIBRARY

Author Statement of Accessibility

Name of Candidate	: Loriza Shazmin Rafi
Degree	: Master of Science in Mathematics
Department/School	: Mathematics/School of Computing, Information & Mathematical Sci
Institution/University	: The University of South Pacific
Thesis Title	: Determination of The Optimal Allocation for Testing Resource of a Modular Software Based on Software Reliability Growth Model
Date of completion of requirements for award	: 12 Mar 2010

1. This thesis may be consulted in the Library without the author's permission. [✓ Yes] / No

2. This thesis may be cited without the author's permission providing it is suitably acknowledged. [✓ Yes] / No

3. This thesis may be photocopied in whole without the author's written permission. [✓ Yes] / No

4. This thesis may be photocopied in proportion without the author's written permission.
   Part that may be copied:

   [✓ Under 10%] ✓ ______ 40-60% ______

   10-20% ______ 60-80% ______

   20-40% ______ Over 80% ______

5. I authorise the University to produce a microfilm or microfiche copy for retention and use in the Library according to rules 1-4 above (for security and preservation purposes mainly). [✓ Yes] / No

6. I authorise the Library to retain a copy of this thesis in e-format for archival and preservation purposes. [✓ Yes] / No

7. After a period of 5 years from the date of publication, the USP Library may issue the thesis in whole or in part, in photostat or microfilm or e-format or other copying medium, without first seeking the author's written permission. [✓ Yes] / No

8. I authorise the University to make this thesis available on the Internet for access by authorised users. [✓ Yes] / No

Signed: [illegible]

Date: 26/02/2010

Contact Address	Permanent Address
College of Fandation Studies	25 Vuna Rd
Grantham Plaza, Raiwaqa,	Nabua, Suva, Fiji Islands
USP	[illegible]

DETERMINATION OF THE OPTIMAL ALLOCATION FOR TESTING RESOURCE OF A MODULAR SOFTWARE BASED ON SOFTWARE RELIABILITY GROWTH MODEL

By

Loriza Shazmin Rafi

A thesis submitted in a fulfillment of the requirements
for the degree of Master of Science in Mathematics.

School of Computing, Information and Mathematics Sciences,
Faculty of Science, Technology and Environment,
The University of the South Pacific.

October, 2009

DECLARATION

I, hereby declare that the work presented in thesis is, to the best of my knowledge, original, and has not been submitted for the award of any other degree at any institution, except where due acknowledgement is made in the text. Information obtained from any published or unpublished sources have been clearly referenced.

[illegible] Date 26/02/2010

Loriza Shazmin Rafi
(S00007105)

The research in this thesis was performed under our supervision and to our knowledge is the sole work of Ms. Loriza Shazmin Rafi.

Supervisors:

[illegible] Date 26.02.10

Dr. Nesar Ahmad
University Department of Statistics & Computer Applications,
T. M. Bhagalpur University,
Bhagalpur 812 007,
Bihar, India.

[illegible] Date 26.02.2010

Dr. M.G.M. Khan
Mathematics and Stats Division Coordinator
School of Computing, Information and Mathematical Sciences,
Faculty of Science, Technology and Environment,
The University of the South Pacific.

DEDICATION

I would like to dedicate this thesis to my husband Mohammed Riaz Ali for his support and encouragement and motivation throughout this research.

TABLE OF CONTENTS

Abstract iv

Acknowledgement v

Publications vi

Preface vii

List of Acronyms x

Chapter 1: Introduction 1

1.1 Software Development 2

1.2 Software Reliability 2

1.2.1 Fault and Failure 3
1.2.2 Failure Intensity 3
1.2.3 Mean Time to Failure 4
1.2.4 Failure Rate 4
1.2.5 Time Index 4

1.3 Non-Homogeneous Poisson Process 5

1.3.1 Introduction 5
1.3.2 Assumptions 6
1.3.3 NHPP Software Reliability Growth Model 7
1.3.4 Musa's Basic Execution Time Model 7
1.3.5 Goel and Okumoto Model 8
1.3.6 S-Shaped Model 9
1.3.7 Inflection S-shaped Model 10
1.3.8 SRGM with Testing-Effort Function 11

1.3.8.1 Exponential Testing-Effort Function 12
1.3.8.2 Weibull-Type Testing-Effort Function 13
1.3.8.3 Logistic Testing-Effort Function 13
1.3.8.4 Log-Logistic Testing-Effort Function 14

1.3.9 Imperfect Debugging 15

1.4 Techniques of Parameter Estimation 15

1.4.1 Maximum Likelihood Estimation 16

1.4.2 Least Square Estimation 16

1.4.3 Weighted Least Square Estimation 17

1.5 Dynamic Programming 18

1.6 Resource Allocation Problems 20

Chapter 2: Literature Survey 21

Chapter 3: A study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models With Imperfect Debugging 31

3.1 Introduction 31

3.2 Review of Exponentiated Weibull TEF 32

3.3 Software Reliability Growth Models 34

3.4 Inflection S-Shaped SRGM with EW Testing-Effort Function 34

3.5 Estimation of Model Parameters 40

3.5.1 Weighted Least Square Estimation 40

3.5.2 Maximum Likelihood Estimation 42

3.6 Data Analysis and Model Comparison 47

3.6.1 Comparison Criteria 47

3.6.2 Model Comparison with Actual data Set 49

3.7 Imperfect Debugging Model 58

3.7.1 Numerical Example 63

Chapter 4: Analysis of an Inflection S-Shaped Software Reliability Model Considering Log-Logistic testing-Effort With Imperfect Debugging 66

4.1 Introduction 66

4.2 Software Reliability Growth Model with TEF 67

4.3 Log-Logistic Testing-Effort Function 67

4.4 Inflection S-Shaped SRGM with Log-Logistic TEF 68

4.5 Imperfect Software Debugging Model 71

4.6 Estimation of Model Parameters 72

4.6.1 Least Square Estimation 73

4.6.2 Maximum Likelihood Estimation 75

4.7 Data Analysis and Model Comparison 79

4.7.1 Comparison Criteria 79

4.7.2 Model Comparison with Actual data Set 81

4.7.3 Numerical Example in an Imperfect Debugging Environment 90

Chapter 5: Determining the Optimal Allocation of Testing Resource for Modular Software System using Dynamic Programming 92

5.1 Introduction 92

5.2 Problem of Testing Resource Allocation 93

5.2.1 Formulation of Resource Allocation Problems as NLPPs 94

5.3 Solution Procedure 99

5.4 Numerical Example and Experimental Results 103

5.5 Discussion 108

Chapter 6: Conclusion 111

References 113

Appendix A 123

ABSTRACT

Software reliability is major concern in the software development process, as unreliable software can cause a failure in the computer system that can be hazardous. A way to enhance the reliability of software is to detect and remove the faults during the testing phase, which begins with module testing whereby, modules, are tested independently to remove substantial amount of faults within the limited resources.

In this thesis, an inflection S-shaped software reliability growth model (SRGM) based on the non-homogeneous Poisson process (NHPP) which incorporates the Exponentiated Weibull (EW) and Log-Logistic testing-effort functions (TEF) is investigated. The weighted least square estimation (WLSE) and least square estimation (LSE) is used to estimate TEF parameters and SRGM parameters are estimated by maximum likelihood estimation (MLE). The performance of both proposed SRGM are demonstrated by using actual data sets from three software projects. Experimental results are compared with the other existing SRGM to show that the proposed models give fairly better predictions. It is shown that, the EW and Log-Logistic TEF are suitable for incorporating into inflection S-shaped NHPP growth models. In addition, the proposed models are also discussed under imperfect debugging environment.

Furthermore, four problems of optimal resource allocation to modules during testing phase are studied. These optimization problems are formulated as nonlinear programming problems (NLPP), which are modeled by the inflection S-shaped software reliability growth models based on a non-homogeneous Poisson process (NHPP) which incorporated the Exponentiated Weibull (EW) testing-effort functions. A solution procedure is then developed using dynamic programming technique to solve the NLPPs.

Finally, numerical examples are given to illustrate the procedure developed in the thesis. In addition, the results are compared with that of Kapur et al. (2004). It is shown that the proposed dynamic programming method for testing resource allocation problem yields a gain in efficiency over Kapur et al.

ACKNOWLEDGEMENT

I would like to express my heartfelt of appreciation to all the people who have guided and contributed to the successful completion of this thesis.

First I would like to thanks my supervisors Dr. Nesar Ahmad and Dr. Mohammed Golam Mostafa Khan of the School of Computing, Information and Mathematical Sciences at the University of the South Pacific for suggesting this project and for their continuance guidance and advice throughout the research.

Special thanks to the Head of School of Computing, Information and Mathematical Sciences, Dr. Jito Vanualaialai, for the approval of the thesis.

I would like to express my gratitude to the University Research Committee for giving the grant and to The University of the South Pacific for financially sponsoring my education without which the completion of the project would have been difficult.

I am very grateful to the Director of College of Foundation Studies, Emily Helen Moala Pouvalu, for letting me to upgrade my qualification while being the full time staff at College of Foundation Studies.

My sincere thank goes to Dr. Zhenquen Li and Mr. Krishna Raghuwaiya, of School of Computing, Information and Mathematical Sciences, USP for their help, regardless of their busy schedule.

I would also like to thank my friends and colleagues at College of Foundation Studies for their constant encouragement and motivation in completing my thesis.

Finally, my humble appreciation to my loving Husband, Mohammed Riaz Ali for his valuable advice and encouragement, which motivated me, believe in myself and aim for higher and making it possible for me to complete my research. Thank you all once again.

PUBLICATIONS

Part of the work described in this thesis is been published in journal papers and conference proceeding as:

1. Ahmad, N., Khan, M. G. M and Rafi, L. S. (2010), “A Study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models with Imperfect Debugging”, *International Journal of Quality and Reliability Management,* Vol. 27, No.1, pp. 89 – 110.

2. Khan, M.G.M., Ahmad, N. and Rafi, L.S. (2008), “Optimal Testing Resource Allocation for Modular Software based on a Software Reliability Growth Model: a Dynamic Programming Approach”, *IEEE Proceedings of 2008 International Conference on Computer Science and Software Engineering*, IEEE Computer Society, Vol. 2, pp. 759 – 762.

3. Ahmad, N., Khan, M.G.M., and Rafi, L.S. (2008), “Inflection S-Shaped Software Reliability Growth Models with Testing-Effort Functions”, *Proceeding of the VI International Symposium on Optimization and Statistics,* Dec. 29 – 31, Aligarh, India.

4. Khan, M.G.M., Ahmad, N., and Rafi, L.S. (2009), “Determining the Optimal Allocation of testing Resource for Modular Software System using Dynamic Programming”, submitted for publication.

PREFACE

This thesis entitled “determination of the optimal allocation for testing resource of modular software based on software reliability growth model” is submitted to the University of the South Pacific, Suva, Fiji to supplicate the Master of Science in Mathematics.

As the society grows in complexity, so do the critical reliability challenges and problems that must be solved. The area of reliability engineering currently received a tremendous attention from numerous researchers and practitioners as well.

Reliability is one of the most important quality attributes of commercial software since it quantifies software failures during the development process. In order to increase the reliability, we should have a comprehensive test plan that ensures all requirements are included and tested. In practice, software testing must be completed within a limited time and project managers should know how to allocate the specified testing-resources among all the modules.

The system reliability depends on both hardware and software reliabilities. The theory of hardware reliability has a long history and was established to improve hardware reliability greatly while the size and complexity of software applications have increased (Xie, 1991).

The topics covered are organized as follows:

- **Chapter 1** gives a brief introduction to software reliability, software development process, basic concepts of software reliability, software reliability models for the failure phenomenon based on the NHPP, imperfect debugging, techniques of estimating testing-effort and SRGM parameters, dynamic programming technique and resource allocation problems.

- **Chapter 2** provides the literature available in the area of software reliability engineering.

- **Chapter 3** presents the study of testing-effort dependent inflection S-shaped software reliability growth model (SRGM) with imperfect debugging. This chapter gives the review of the Exponentiated Weibull testing effort, mathematical description of the SRGM used. The estimation of model parameters using MLE and WLSE methods, the analysis of the three actual data, the comparison criteria, comparison of the SRGM model proposed in this thesis with other existing models. Finally, the proposed model is discussed under the imperfect debugging environment. This chapter is based on my joint paper entitled, "Inflection S-Shaped Software Reliability Growth Models with Testing-effort Function", presented in the VI International Symposium on Optimization and Statistics, Dec 29-31, Aligarh, India and "A Study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models with Imperfect Debugging", International Journal of Quality and Reliability Management, Vol. 27, No.1, pp. 89 – 110.

- **Chapter 4** presents the analysis of inflection S-shaped SRGM considering Log-Logistic testing-effort with imperfect debugging. This chapter presents the Log-Logistic testing effort function, mathematical description of the Software Reliability Growth model used, the model parameters estimation techniques using the MLE and LSE methods, the comparison criteria, validation of the proposed model using three actual data, comparison of the SRGM model developed in this thesis with other existing models. Finally, the developed model is discussed under the imperfect debugging environment.

- **Chapter 5** discusses problems of testing resource allocation and the details of the formulation of the problems as NLPPs are provided. The solution procedure for the problems using dynamic programming technique is discussed. The computational details of the solution procedure are illustrated with numerical examples. Finally, an investigation is carried out to compare the results obtained by the Kapur et al.

(2004) formulations using dynamic programming approach. This chapter is based on my joint paper entitled, “Optimal Testing Resource Allocation for Modular Software based on a Software Reliability Growth Model: a Dynamic Programming Approach”, IEEE Proceedings of 2008 International Conference on Computer Science and Software Engineering, IEEE Computer Society, Vol. 2, pp. 759 – 762.

- **Chapter 6** gives a brief conclusion. A comprehensive list of references is presented at the end of the thesis.

- **Appendix A** contains the algorithm for solving NLPP developed in chapter 5.

LIST OF ACRONYMS

SRGM	-	Software Reliability Growth Model
NHPP	-	Non-Homogeneous Poisson Process
MTTF	-	Mean Time to Failure
MTBF	-	Mean Time between Failures
TEF	-	Testing-Effort Function
LSE	-	Least Square Estimation
WLSE	-	Weighted Least Square Estimation
MLE	-	Maximum Likelihood Estimation
EW	-	Exponentiated Weibull
AE	-	Accuracy of Estimation
MSE	-	Mean of Squared Errors
NLPP	-	Non-Linear Programming Problem

CHAPTER 1

Introduction

In today's hi-tech world nearly everyone depends upon the continued functioning of a wide array of complex machinery and equipment for our everyday safety, security, mobility and economic welfare. We expect our electric appliances, lights, hospital monitoring control, next-generation aircraft, nuclear power plants, data exchange systems, and aerospace applications, to function whenever we need them. When they fail, the results can be catastrophic, injury or even loss of life. Thus, how to measure and predict reliability of software has lead to a great demand for high quality software products. However, poor performance due to unreliable software is exhibited by many systems. Therefore, systems must be well documented, tested and verified to ensure maximum performance as designed.

To improve software quality, software reliability plays an important role in many aspects throughout the software life cycle. In an effort to address the continuing demand for high quality software, a colossal amount of software reliability growth models have been proposed in recent years. In spite of diversity and elegance of many of these, there is still a need for models which can be more readily applied in practice. One aspect which should perhaps be taken into account of is the complex and challenging nature of the testing of software. Testing is the only effective way of conforming to high software reliability and one of the five important stages in software development.

A faulty program sometimes can still give correct output, thus reliability of software is not deterministic. Reliability is therefore measured probabilistically. Measuring software reliability alone does not solve the problem of achieving reliable software; it just reflects the quality of software on hand. Testing is usually a lengthy process in the software industry accounting for 40-50% of the development process (Musa *et al*. 1987). The faults detected as time elapses is used to update the reliability information of the software. This information could be translated into determining the testing time or resources required in order to meet various criterions of reliability or cost. It is estimated that only 20% of software developed is used (Musa *et al*. 1987). Also, the resource allocated to software testing is usually limited. Therefore, the software testing process must be efficient and cost effective. The reliability of the software is usually estimated by devising mathematical models describing a typical behavior of debugging process.

1.1 Software Development

Software development life cycle is divided into various phases such as specification phase, design phase, coding phase, testing phase, and operational and maintenance phase (Musa *et al*.1987)**.** The initial stage of the software life cycle does not have the failure data therefore, a predictive model is needed. This type of model predicts the number of faults in the program before testing. In the testing phase software goes through debugging process whereby the reliability of the software is improved.

Thus, to achieve the objective reliability level, a reliability growth model is needed to estimate the current reliability level and the time and required resources. During this phase, reliability estimation is based on the analysis of failure data. The main goal of software testing is to eliminate faults and obtain reliable software. If the total number of faults in software is known beforehand then the number of faults eliminated would indicate the system reliability.

1.2 Software Reliability

Software reliability is defined as the probability that the software will be functioning without failure under a given environmental condition during a specified period of time (Xie, 1991). The software reliability is the most dynamic attribute (metric) which can measure and predict the operational quality of the software system (Xie, 1991). Software is considered to have performed a successful operation, when it functions completely as expected, without any

failure. Software that fails less often is considered to have higher quality than software that fails frequently. For accurate measurement during test, runs should be selected randomly with the same probability expected to occur in operation. Therefore reliability is best measured probabilistically. Reliability represents a user-oriented view of software quality.
Mathematically, reliability $R(t)$ is the probability that a system will be successful in the interval from time 0 to time *t*.

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

where *T* is random variable denoting the time-to-failure time.

It is important to understand the basic concepts of software reliability modeling before specific models or practical applications can be addressed. Some of the concepts are discussed below.

1.2.1 Fault and Failure

There have been different terminologies used by different authors to describe the software reliability problems such as fault, error, failure, bug, defects, etc. If for some input data, the output result is incorrect then software is said to contain a *fault.* Fault has always been an existing part in the software and by correcting the erroneous part of the software, it can be removed. In general all those parts of the software which may cause any problems is regarded as software faults (Xie, 1991).
Failure as defined by Musa *et al*. (1987) is the departure of external results of program operation from program requirements on a run. A departure causes the difference between the desired output value specified by the requirements for a particular run and the actual output value.

1.2.2 Failure Intensity

Failure intensity is an alternative way of expressing reliability. The failure intensity function is the rate of change of the mean value function, $m(t)$ or the number of failures per unit time.

Mean value function represents the average cumulative failures associated with each time point. The failure intensity function is obtained by:

$$\lambda(t) = \frac{dm(t)}{d(t)} .$$

1.2.3 Mean Time to Failure

Mean time to failure is the average value of the next failure interval. MTTF is the definite integral evaluation of the reliability function.

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

The larger MTTF indicates better reliability (Musa, 1987). The MTTF is used when the failure time distribution function is specified because the reliability level implied by the MTTF depends on the fundamental failure time distribution.

1.2.4 Failure Rate

Failure rate is the rate at which failures occur in a certain time interval [$t_{1,}$ t_2]. It is the probability that a failure per unit time occurs in the interval, given that a failure has not occurred prior to t_1, the beginning of the interval. Failure rate is a function of a time. Thus, it is given as

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

1.2.5 Time Index

A Software Reliability Growth Model (SRGM) is one of the fundamental techniques to assess software reliability quantitatively. For software reliability modeling, the span of the time may be considered as calendar time, clock time and execution time (Huang *et al*. 2000).

The *execution time* for a program is the time that is actually spent by a processor in executing the instructions of that program. The *clock time* is the elapsed time from start to end of program execution on a running computer. The *calendar time* is the familiar garden variety of time that is normally experienced.

1.3 Non-Homogeneous Poisson Process (NHPP) Models

1.3.1 Introduction

Non-Homogeneous Poisson process (NHPP) is a well developed stochastic process has been successfully used in the reliability study of software system. A NHPP is a realistic model for predicting software reliability and has a very interesting and useful interpretation in debugging and testing the software. Non-Homogeneous indicates that the characteristics of the probability distributions that make up the random process vary with time. Poisson simply refers to the probability distribution of the value of the process at each point in time (Runarsson, 2004). NHPP models are especially useful to describe failure processes which possess certain trends such as reliability growth or deterioration.

One of the first NHPP models is suggested by Schneidewind (1975). The most well- known NHPP model is the model studied in Goel and Okumoto (1979) which later on, has been further generalized and modified by various authors in order to improve the goodness-of-fit to real software failure data (Yamada *et al*. 1986; 1987; 1990; 1993; Kapur *et al*. 1994; Kapur and Younnes, 1996; Huang *et al*. 1997; 1999; 2000; Kuo *et al*. 2001; Huang and Kuo, 2002; Huang, 2005; 2005a; Bokhari and Ahmad, 2006; Ahmad *et al*. 2008; Quadri *et al*. 2008; 2008a).

The cumulative number of software failures up to time $t, N(t)$, can be described by a NHPP. For the counting process $\{ N(t), t > 0 \}$ modelled by NHPP, $N(t)$ follows a Poisson distribution with parameter $m(t)$, which is called the mean value function. The function $m(t)$ describes the expected cumulative number of failures in [0, t). Hence $m(t)$ is a very useful descriptive measure of the failure behavior.

1.3.2 Assumptions

The NHPP model has the following assumptions:

- The failure process has an independent increment i.e., the number of failures occurred during the time interval $(t,\ t+\Delta t]$ depends on current time t and the length of time intervals Δt, and does not depend on the past history of the process (Xie, 1991).

- The failure rate of the process is

$$\text{P}\{\text{exactly 1 failure in } (t, t+\Delta t) = \text{P}\{N(t+\Delta t) - N(t) = 1\}\}$$

$$= \lambda(t)\Delta t + o(\Delta t),$$

  where $\lambda(t)$ is the instantaneous failure intensity (Xie, 1991).

- During a small interval Δt, the probability of more than one failure is negligible, i.e.,

$$\text{P}\{\text{2 or more failures in } (t, t+\Delta t)\} = o(\Delta t).$$

- Initial condition is $N(0) = 0$.

Based on the above assumptions, it can be shown that $N(t)$ has a Poisson distribution with *m(t)*, i.e.,

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

The mean value functions of the cumulative number of failures, *m(t),* can be expressed in terms of the failure rate of the program, i.e.,

$$m(t) = \int_0^t \lambda(s)ds.$$

Inversely, knowing *m(t),* the failure intensity at time *t* can be obtained as

$$\lambda(t) = \frac{dm(t)}{dt}.$$

- Generally, by using different non decreasing functions *m(t),* different NHPP models can be obtained. Many NHPP models have been studied as mentioned earlier due to great variability of the mean value functions (Xie, 1991).

1.3.3 NHPP Software Reliability Growth Models

A software reliability growth model (SRGM) is a mathematical expression of the software fault occurrence and the removal process which explains the time dependent behavior of fault removal. Software reliability is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment (Musa *et al*. 1987). The Poisson Process SRGMs provides an analytical framework for describing the software failure phenomenon during testing.

Hence, accurately modeling software reliability and predicting its possible trends are essential for determining overall product's reliability. Numerous SRGMs have been developed during the last three decades and they can provide very useful information about how to improve reliability (Musa *et al*. 1987; Xie, 1991; Lyu, 1996; Pham, 2000). Some important metrics, such as the number of initial faults, failure intensity, reliability within a specific time period, number of remaining faults, mean time between failures (MTBF), and mean time to failure (MTTF), can be easily determined through SRGMs. The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time. Some of the models included in this group are given below.

1.3.4 Musa's basic execution time model

The basic execution time model can be regarded as variant of the exponential growth models. Musa's model is known to be the most practical model of all (Xie, 1991). It has made a unique contribution in the understanding of the relationship between execution time and calendar time. The parameters of the model have explicit physical interpretation that allows predicting a future reliability figure of a program based on the failure detections process of a test. The model is given by the function:

$$m(t) = a \ \ 1 - e^{-\phi\beta t} \ \ .$$

where *t* is the execution time, that is, the total CPU time utilized to complete the test case runs up to a time of observation, *a* is the number of failures in the program, β is the fault detection factor, *m(t)* is the total failures that would be discovered as a result of test case runs up to the time of observations, and ϕ is the constant per-fault hazard rate.

By differentiating above *m(t)* with respect to *t*, the following expression is obtained for the failure intensity rate $\lambda(t)$:

$$\lambda(t) = a\phi\beta e^{-\phi\beta t} .$$

1.3.5 Goel and Okumoto Model

The Goel and Okumoto model (G-O model) is also regarded as a variant of the exponential growth model. In fact the Musa basic execution and G-O model are mathematically isomorphic (Ohba, 1984). Goel and Okumoto presented a simple model for the description of software failure process by assuming that the cumulative failure process is NHPP with a simple mean value function.
The assumptions of the G-O model are (Xie, 1991):

1. The cumulative number of faults detected at time *t* follows a Poisson distribution.
2. All faults are independent and have the same chances of being detected.
3. All detected faults are removed immediately and no new faults are introduced.

The mean value function of the Goel and Okumoto NHPP model is given by:

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

where t is the time of observation, a and b are parameters to be determined using collected failure data and *m(t)* is the number of failures detected up to the time of observation.

The failure intensity rate function $\lambda(t)$ is given by:

$$\lambda(t) = \frac{dm(t)}{dt} = abe^{-bt}\ .$$

1.3.6 S-shaped Model

The S-shapedness is generally elucidated by the fact that faults are neither of the same size nor independent. There are some faults covered by the other faults and unless these faults are detected and removed, the covered faults cannot be detected (Xie, 1991).

The NHPP S-shaped model is based on the following assumptions:

1. The fault detection rate differs among faults.
2. Each time a software failure occurs, the software fault which caused it is immediately removed, and no new faults are introduced.

This is described by the following differential equation:

$$\frac{\partial m(t)}{\partial t} = b(t)\ \ a - m(t)\ \ ,$$

where

a	=	expected total number of faults that exist in the software before testing.
$b(t)$	=	failure detection rate, also called the failure intensity of a fault.

$m(t)$ = expected number of failures detected at time *t*.

To solve the above differential equation, method of separable variable is used. Integrating the differential equation with the initial condition at $t = 0,\ m(t) = 0$, gives:

$$\int_0^t \frac{dm(u)}{a - m(u)} = \int_0^t b(u)dt$$

$$-\ \ln(a - m)\Big|_0^t = \int_0^t b(u)du$$

$$\ln\left(\frac{a - m(t)}{a}\right) = -\int_0^t b(u)du$$

Applying exponential both sides give:

$$\frac{a - m(t)}{a} = e^{-\int_0^t b(u)du}$$

Thus solve for *m(t)*

$$a - ae^{-\int_0^t b(u)du} = m(t)$$

$$m(t) = a[1 - e^{-\int_0^t b(u)du}].$$

Several different S-shaped NHPP models have been proposed in the existing literature, but the most interesting ones are the delayed S-shaped and inflection S-shaped NHPP model (Xie, 1991). This is because these models were able to successfully fit a given data set when others couldn't and excellent fits and predictions can be obtained due to presence of S-shaped behavior in the data set (Lyu, 1996).

1.3.7 Inflection S-shaped Model

Ohba (1984) postulated the following assumptions by modeling the inflection S-shaped model based on the dependency of faults (Pham, 2000):

1. Some of the faults are not detectable before some other faults are removed.
2. The probability of failure detection at any time is proportional to the current number of detectable faults in the software.
3. Failure rate of each detectable fault is constant and identical.
4. The isolated faults can be entirely removed.

Assume that

$$b(t) = \frac{b}{1+\beta e^{-bt}},$$

where b and β represent the failure-detection and inflection factor respectively. The mean value function of the inflection S-shaped NHPP model is given by:

$$m(t) = \frac{a(1-e^{-bt})}{1+\beta e^{-bt}}.$$

The failure intensity function is given by:

$$\frac{dm(t)}{dt} = \lambda(t) = \frac{ab(1+\beta)e^{-bt}}{(1+\beta e^{-bt})^2}.$$

In this thesis, the software reliability growth model is modelled by the inflection S-shaped model which is discussed later.

1.3.8 SRGM with Testing-Effort

In the development of software products, the testing phase is an integral but costly part. Resources such as manpower and computer time are consumed during testing. The nature and amount of resources spent determines the identification and removal of faults. The behaviour of the testing resource expenditures over the testing period can be observed as a consumption curve of the testing-effort. The increase in reliability is strongly dependent on the allocated testing-effort. The consumption of testing-effort employed to detect and remove the faults during the testing phase is either homogeneously or non-homogeneously distributed. Software reliability models have been developed earlier by incorporating some testing-effort functions (TEF) (Musa *et al*. 1987; Xie, 1991; Lyu, 1996; Pham, 2000).

It has been assumed in all the SRGM discussed so far that as the testing time increases, testing effort also increases. If testing time becomes quite large, testing effort also becomes quite large. In reality, no software developer may spend infinite resources on testing effort expenditure. Let $w(t)$ be the testing effort expenditure at time t. Then the cumulative testing effort expenditure in $(0,t)$ is expressed as:

$$W(t) = \int_0^t w(x)\,dx.$$

Under the assumption that the number of faults detected in $(t, t+\Delta t)$ per unit testing effort expenditure is proportional to the remaining faults, i.e:

$$\frac{m'(t)}{w(t)} = b(a - m(t)).$$

Solving the above give

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

1.3.8.1. Exponential Testing-Effort Function

Software reliability growth models, incorporating the amount of test-effort spent on software testing was proposed by Yamada *et al*. (1986). They assumed that both the test-effort during the testing and the software development effort can be described by the Rayleigh curve (a curve that yields a good approximation to the actual labour curves on software projects). They also assumed an exponential curve as an alternative to the Rayleigh curve. The exponential curve is often used when the testing-effort is uniformly consumed with respect to the testing time. The cumulative testing-effort consumed in $(0,t]$ is

$$W(t) = \alpha \times (1 - \exp[-\beta t]).$$

The current testing-effort consumption is:

$$w(t) = \alpha\beta \exp[-\beta t]) .$$

1.3.8.2. Weibull -Type Testing-Effort Function

According to Musa (1999), Musa *et al*. (1987), Yamada *et al*. (1986; 1993) and Putnam (1978), testing-effort should not be assumed constant throughout the testing phase. Instantaneous testing-effort ultimately decreases during the testing life- cycle because the cumulative testing-effort approaches a finite limit. Hence, Yamada *et al*. (1986; 1993) show that the testing-effort can be described by a Weibul-type distribution and have the following 3 cases.

1) Exponential Curve: The cumulative testing-effort consumed in (0, t] is

$$W(t) = \alpha.[1 - \exp(-\beta.t)] .$$

2) Raleigh Curve: The cumulative testing-effort consumed is

$$W(t) = \alpha.\left[1 - \exp\left(-\frac{\beta}{2}.t^2\right)\right].$$

3) Weibull Curve: The cumulative testing-effort consumed is

$$W(t) = \alpha.[1 - \exp(-\beta.t^m)] .$$

For the Weibull-type curves, when $m = 1$ or $m = 2$, the result is the exponential or Rayliegh curve respectively; therefore, they are special cases of the Weibull TEF.

1.3.8.3. Logistic Testing-Effort Function

Weibull-type curve can fit the data well under the general software development environment and is widely used in software reliability modeling, it has the apparent peak phenomenon when $m > 3$ [Huang *et al*. (1997), (1999), (2000)]. An alternative is the Logistic testing-effort function (TEF), first presented by Parr (1980). Later on, Huang and Kuo (2002) extended the Yamada works by incorporating logistic testing-effort function into SRGM. This function was fairly accurate as reported by the Yourdon 1978-1980 project survey [DeMarco (1982)]. The cumulative testing-effort consumption in time (0 *t*] is

$$W(t) = \frac{N}{1 + A.\exp(-\alpha.t)}.$$

The current testing effort consumption is

$$w(t) = \frac{dW(t)}{dt} = \frac{N.A.\alpha.\exp(-\alpha.t)}{[1 + A.\exp(-\alpha.t)]^2}$$

$$= \frac{N.A.\alpha}{\left[\exp(\frac{\alpha.t}{2}) + A.\exp(\frac{-\alpha.t}{2})\right]}.$$

1.3.8.4. Log-Logistic Testing-Effort Function

Log-logistic testing-effort can capture the increasing/decreasing nature of the failure occurrence rate per fault according to Gokhale and Trivedi, (1998). Recently, Bokhari and Ahmad (2006) presented how to use the log-logistic curve to describe the time-dependent behavior of testing-effort consumptions. The cumulative log-logistic TEF is given by

$$W(t) = \alpha\left(\frac{\beta t^{\ \delta}}{1 + \ \beta t^{\ \delta}}\right).$$

The current testing effort consumption is given by

$$w(t) = [\alpha\beta\delta(\beta t)^{\delta-1}]/[1 + (\beta t)^{\delta}]^2 .$$

1.3.9 Imperfect Debugging

It is widely recognized that the debugging processes are usually imperfect. Software faults are not completely removed because of the difficulty in locating them or because new faults might be introduced. The testing phase of the software development involves the debugging process. It could be perfect debugging or imperfect debugging. A perfect debugging assumes a fault is certainly removed whenever a failure occurs; the number of remaining faults is a decreasing function of debugging time. Whereas, an imperfect debugging assumes faults may or may not be removed, introduced, or changed at each debugging, the number of remaining faults may increase or decrease (Pham, 2000).

The imperfect debugging phenomenon exists and the probability of imperfect debugging is constant during the testing. The assumption does not reflect the dynamic nature of the testing phase in general and the experience of the testing team does not improve throughout the progress of the test. On the contrary, the experience of the testing team grows with the progress of the test leading to reduction in the probability of imperfect debugging in the later stage of the test (Kapur *et al*. 1999)

1.4 Techniques for Parameter Estimation

The success of a software reliability growth model depends heavily on the quality of the failure data collected. The parameters of the SRGM are estimated based upon this data. In order to validate the proposed model and to compare its performance with other existing models, experiments on actual software failure data is performed. Maximum Likelihood estimation (MLE), Least Square estimation (LSE) and Weighted Least Square estimation (WLSE) techniques are used to estimate the model parameters (Musa *et al*. 1987; Musa, 1999; Lyu, 1996). Least squares and maximum likelihood have been suggested and widely used for estimation of parameters of SRGM. Both the methods are solved using first order differential equation, which involves mathematical computations.

1.4.1 Maximum Likelihood Estimation

Estimation by maximum likelihood is a general technique that may be applied when the underlying distributions of the data are specified or known. This technique is applied for large sample sizes. It can be applied to any problem for which the general form of the joint pdf is known. This method has been employed by many researchers to obtain parameter estimates. Likelihood equations are derived from the model equations and the assumptions which underlie these equations. The parameters are then taken to be those values which maximize these likelihood functions. These values are found by taking the partial derivative of the likelihood function with respect to the model parameters, the maximum likelihood equations, and setting them to zero. Sometimes, however, the likelihood equations may be complicated and difficult to solve explicitly. In that case one may have to resort to some numerical procedure to obtain the estimates (Musa *et al*. 1987). In this manuscript, maximum likelihood estimation is used to estimate the parameters of the SRGM.

The foundation of the maximum likelihood method is the likelihood functions. Lets denote y_k be the number of faults detected in time interval $(0, t_k]$, where (k = 1, 2, …, $n; 0 < t_1 < t_2 < ..... < t_n$). The likelihood function for the NHPP model with mean value function $m(t)$ as follows:

$$L(y_1, y_2, ..., y_k) = \prod_{k=1}^{n} \frac{[m(t_k) - m(t_{k-1})]^{(y_k - y_{k-1})} \exp\ m(t_k) - m(t_{k-1})}{(y_k - y_{k-1})!} \, .$$

The parameters in $m(t)$ can be estimated by maximizing this likelihood function. Usually, numerical procedures have to be used in solving the likelihood equations.

1.4.2 Least-Square Estimation

Least square is a time-honored estimation procedure that has been in use since the early nineteenth century. Like maximum likelihood estimation, it is a fairly general technique which can be applied in most practical situations and is better for small or medium sample sizes (Musa *et al.* 1987). Unlike maximum likelihood, which can be applied to any problem, in least squares the parameters to be estimated must arise in expressions for the means of the observations. When the parameters appear linearly in these expressions then the least squares estimation problem can be solved in closed form. In order to estimate the parameters, the total squared difference between observed and predicted values is minimized.

The testing effort data are given in the form of testing effort $W_k\ (W_1 < W_2 < .... < W_n)$ consumed in time $(0, t_k]$, where k = 1, 2, …, n. Let ($b_1, b_2, b_3, ...$) be any number of parameters in a testing-effort function, then the testing effort parameters are estimated by minimizing

$$S(b_1, b_2, b_3, ...) = \sum_{k=1}^{n} [W_k - W(t_k)]^2 .$$

1.4.3 Weighted Least Square Estimation

The least square estimation method gives equal weight to each data point when a model is fitted. However, software testing practice indicates that the recent data points may influence the reliability projection more than earlier data points. Therefore, it is desirable to make the model fit better to the later data points, so that better predictions can be obtained for the future points in time.

A commonly occurring case is that of weighted least squares, when the observations are assumed uncorrelated, but with different variances. The parameters are obtained by minimizing

$$\mathrm{S}(b_1, b_2, b_3, ...) = \sum_{k=1}^{n} l_k \ W_k - W(t_k) \ ^2 ,$$

where l_k is weight assigned to data points according to their proper amount of influence over the parameter estimates.

The standard Linear Least Squares paradigm may be extended to accommodate situations where:

◊ Different weights are assigned to different observations to account for their relative importance.

◊ The measurement faults on the y^i are correlated.

◊ The noise variance is not the same throughout the predictor space (heteroscedasticity).

◊ The mean of a population has to be estimated from several samples with different sizes.

1.5 Dynamic Programming

Many decision-making problems take place in several stages. The problems in which decisions are to be made sequentially at different stages of solution are called multistage decision problems. Many multistage decision problems can be formulated as a mathematical programming problem. The dynamic programming technique, developed by Richard Bellman in early 1950, is a computational method which is well suited for solving MPPs that may be treated as a multistage decision problem in which a sequence of interrelated decision has to be made.

The general nature of the MPPs that can be solved by dynamic programming technique may be described as follows:

1. The given MPP may be described as a multistage decision problem, where at each stage the value(s) of one or more decision variables are to be determined.

2. The problem must be defined for any number of stages and have the same structure irrespective of the number of stages.

3. At each stage, there must be a specified set of parameters describing the state of the system, that is parameters on which the values of the decision variable and the objective function depends.

4. The same set of state parameters must be described as the state of the system irrespective of the number of the stages.

5. The decision at any stage, that is, the determination of the decision variable(s) at any stage must have no effect on the decisions of the remaining stages except in changing the values of the parameters which describe the state of the system.

When the above conditions are fulfilled, then the given MPP can be solved by dynamic programming technique. This technique decomposes the original problem of n-variables into n-sub problems (called stages) of single variable. The solution of a problem is achieved in a sequential manner starting from one stage problem, moving onto a two stage problem, to a three stage problem and so on until finally all stages are included. The solution for n stages is obtained by adding the n^{th} stage to the solution of $n - 1$ stages.

This could be done by defining a relation between the solutions of the two adjacent stages. This relation is known as the *Recurrence Relation of Dynamic Programming*.

The basic concept of dynamic programming is contained in the principle of optimality proclaimed by Richard Bellman. According to him,

"An optimum policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision".

The principle of optimality implies that given the initial state of a system, an optimal policy for the subsequent stages does not depend upon the policy adopted at the preceding stages. That is, the effect of a current policy decision on any of the policy decisions of the preceding stages need not be taken into account at all. It is usually referred to as the Markovian property of dynamic programming.

As described elsewhere in this manuscript, the optimization problems in determining the optimal allocation for the testing resource of modular software subject to different constraints are mathematical programming problems having the sum of fractional functions as the common objective. These are solved using a dynamic programming approach. Hence, dynamic programming technique may be used as a tool to solve the MPPs.

1.6 Resource Allocation problems

Consider a software having N modules that are being tested independently for removing faults lying dormant in them. The duration of module testing is often fixed when scheduling is done for the whole testing phase. Hence, limited resources are available, which need to be allocated judiciously. If m_i faults are expected to be removed from the ith module with effort X_i, the resulting testing resource allocation problem can be stated as follows:

$$\text{Maximize } \sum_{i=1}^{N} m_i$$

$$\text{Subject to } \sum_{i=1}^{N} X_i \le Z \quad , \qquad X_i \ge 0, \; i = 1, \ldots, N \, .$$

The above optimization problem is the simplest one as it considers the resource constraint only. Later in this manuscript, additional constraints are incorporated to the basic model. Dynamic programming technique is used to solve this optimization problem as mentioned earlier.

CHAPTER 2

Literature Survey

Since the early 1970's, the development of software reliability made its greatest impact in which a number of SRGM have been proposed for the estimation of reliability growth of products (Lyu, 1996; Xie, 1991; Musa *et al*. 1987). The SRGM proposed were the works of Jelinski and Moranda (1972), Shooman (1972, 1973, 1976, and 1970) and Cautinho (1973). Since then more than 100 other models and modifications have been proposed in the literature, and software reliability continues to be a very active area for research amongst engineers, mathematicians and statisticians.

Shooman (1972) and Jelinski and Moranda (1972) introduced a simple model for the estimation of the number of initial software faults, using failure data collected during software testing phase. Schick and Wolverton (1973) proposed a similar model to that of Jelinski and Moranda except used Rayleigh distribution in modeling time between failures whereby the hazard rate was proportional to the number of faults remaining and power of the time.

Also, several SRGMs had been developed in the literature assuming the debugging process to be perfect and thus implying that there is one-to-one correspondence between the number of failures observed and fault removed. Goel and Okumoto (1978) developed a modification of the Jelinski and Moranda model for the case of imperfect debugging. Schneidwind (1975) viewed fault detections per time interval as a Non-Homogeneous Poisson Process (NHPP) with an exponential mean value function. Goel and Okumoto (1979), described failure detection as a Poisson process with an exponentially decaying rate function. Also a simple modified NHPP model was investigated by Yamada, Ohba, and Osaki (1983), where the cumulative number of failures detected is described by S-Shaped curve.

The term Non-Homogeneous indicates that the characteristics of the probability distributions that make up the random process vary with time. The term Poisson simply refers to the probability distribution of the value of the process at each point in time [Runarsson, 2004].

Musa (1975) proposed an execution time model. His findings were that software reliability theory should be based on execution time rather than calendar time as calendar time does not account for varying usage of the program in either test or operation.

Several SRGM which relate the number of failures (faults identified) and execution time (CPU time/Calendar time) have been discussed by Goel and Okumoto (1979), Ohba (1984), Yamada *et al*. (1985, 1986), Musa *et al*. (1987), Xie (1991), Kapur *et al*. (2004).

Some of these models developed were as calendar time models, execution time models or is not explicitly specified the type of time being used. Non-Homogeneous Poisson Process (NHPP) as a stochastic process has been successfully used in the reliability of software system. Stochastic Process is defined (Musa *et al*. 1987) as the situation in which observations are made over a period of time and are influenced by change or random effects, not just at a single instant but throughout the entire interval.

Later, the idea of incorporating testing-effort into the modeling process was introduced. Very few Software Reliability Growth Models have been developed which define explicitly the testing-effort functions during testing. These SRGMs have been developed based on Non-Homogeneous Poisson Process (NHPP). The SRGMs based on NHPP for software testing incorporated the testing effort functions such as Exponential, Rayleigh, Weibull, Exponentiated Weibull, Logistic, Generalized Logistic, Log-Logistic and Burr type X (Musa, 1987; Yamada *et al*. 1993; Kapur *et al*. 2004; and Huang *et al*. 2000; 2001; 2002; Bokahri and Ahmad, 2006; Ahmad *et al*. 2008; 2008a; 2009).

Most software developers and managers always want to know the date on which the desired reliability goal will be met. Also, software developers want to detect more faults in practice, therefore, it is advisable to introduce new techniques, tools, or consultants, et cetera. Hence, researches had been done on optimal software release policies and include reliability in the cost function. A considerable amount of testing resources is required during software module testing. Researches based on optimal resource allocation had been done based on both Non-Homogeneous Poisson Process and Hyper-Geometric Distribution Model. Hyper-geometric distribution software reliability growth model was developed for estimating the number of software faults initially in a program.

Some of the works of the authors in the field of Software Reliability is mentioned below:

Yamada *et al*. (1986) developed a realistic software reliability growth models incorporating the effect of testing-effort. The software fault detection phenomenon in software testing was modeled by a Non-Homogeneous Poisson Process. The software reliability assessment measures and the parameters were investigated by the estimation methods of maximum likelihood estimators and least square estimators. The testing-effort expenditures were described by exponential and Rayleigh curves.

Ohtera and Yamada (1990) presented a software reliability growth model based on a NHPP. The model described the time-dependent behavior of software faults detected and testing resource expenditures spent during the testing. They discussed the optimal allocation and control of testing resources among software modules which could improve reliability and shorten the testing stage. Based on the model, numerical examples of these two software testing management problems were presented.

Bhalla (1992) discussed software release policies for a flexible reliability growth model based on a NHPP. The optimization criterion was to minimize the cost subject to achieving a given level of reliability. Numerical results were presented. The author claimed that the flexible model discussed provided a framework within which the release policies for many SRGMs could be described.

Yamada *et al*. (1993) developed a software reliability growth model incorporating the amount of the test-effort expanded during the software testing phase. The time-dependent behavior of test-effort expenditures was described by a Weibull curve. The model was formulated by a non homogeneous Poisson process. Using the model, the method of the data analysis was developed. The reliability growth parameters were estimated by the maximum likelihood estimation method.

Zeephongsekul (1996) proposed a software reliability growth model which incorporated imperfect debugging and generation of fault. He showed that its trend curve exhibits either an exponential or S-shaped growth depending on a simple condition involving the relative efficiency in fault detection between the original faults and faults generated as a consequences of imperfect debugging.

Kapur and Younes (1996) developed a model which described imperfect debugging process and had the inbuilt flexibility of capturing a wide class of growth curves. The relevance of the model had been shown on several data sets obtained from different software development projects.

Hou *et al*. (1996) investigated two optimal resource allocation problems in software module testing based on the Hyper-Geometric Distribution Model (HGDM) software reliability growth model. Firstly, the minimization of the number of software faults still undetected in the system after testing given a total amount of testing resources and secondly, the minimization of the total amount of testing resources repaired, given the number of software faults still undetected in the system after testing. Moreover, they introduced two simple allocation methods based on the concept of “average allocation” and “proportional allocation”. According to their experimental results, it was shown that the optimal allocation method could improve the quality and reliability of the software system much more significantly than these simple allocation methods.

Leung (1997) also studied a dynamic resource allocation strategy for software module testing. The strategy took into account the variations of the number of detected faults during testing, re-estimated the model parameters using all the available fault detection data and dynamically allocated the testing resources to the software modules. The simulation result showed that this strategy could effectively reduce the variance of the number of remaining faults.

Huang *et al*. (1997; 2002) investigated a software reliability growth model based on NHPP which incorporated Logistic testing effort function. They showed that a logistic testing effort function could be expressed as a software development/test effort curve and gives a reasonable predictive capability for the real failure data. The parameters were estimated and experiments on three actual test/debug data sets were illustrated. Their result showed that the software reliability growth model with logistic testing effort function could estimate the number of initial faults better than the model with Weibull-type consumption curve. They also discussed the optimal release policy on this model based on cost-reliability criterion.

Gokhale and Trivedi (1998) proposed the Log-Logistic software reliability growth model, the development of which was primarily motivated due to the inadequacy of the existing models to exhibit either constant, monotonic increasing or monotonic decreasing failure occurrence rates per fault and describe the failure process underlying certain failure data sets. Equations to obtain the maximum likelihood estimates of the parameters of the existing finite failure NHPP models, as well as the log-logistic model based on times between failures data were developed. The analysis of two failure data sets which led then to the log-logistic model using arithmetic and Laplace trend tests, goodness-of-fit test, bias and bias trend tests was presented.

Huang and Lyu (1999) presented two important issues on software reliability modeling and software reliability economics: testing effort and efficiency. They extended the logistic testing-effort function into a general form because according to them the Generalized Logistic testing effort function could be used to describe the actual consumption of resources during software development process. Moreover, they incorporated the generalized Logistic testing effort function into software reliability modeling and its fault prediction capability was evaluated through four numerical experiments on real data. They also addressed the effects of automated techniques on increasing the efficiency of software testing.

Pham *et al*. (1999) developed a model and compared the descriptive and predictive ability with a set of classical NHPP reliability models. The experiment included both imperfect debugging and a time-dependent fault-detection rate into an NHPP software reliability growth model (SRGM). Incorporating the above showed the improvement in both descriptive and the predictive properties of a model and is worth the extra model-complexity and the increased number of parameters required for a better relative fit.

Huang *et al*. (2000) demonstrated that the Logistic testing effort function is practically acceptable/helpful for modeling software reliability growth and providing a reasonable description of resource consumption. They incorporated the logistic testing effort function into S-shaped model for further analysis. To demonstrate the estimation procedures and results, a realistic failure data set is used in the experiments. Furthermore, the analysis of the proposed model under the imperfect debugging environment was investigated.

Xie and Yang (2001) studied the problem of optimal testing-time allocation for modular software systems. A generic formulation of the problem was presented to illustrate the optimization algorithm and the solution. In addition, an example of sensitivity analysis was shown as software reliability growth models consisting of a number of parameters.

Kuo *et al*. (2001) and Huang (2005a) proposed a SRGM based on NHPP with the central focus to provide an efficient parametric decomposition method for software reliability modeling. In doing so, Kuo et al. (2001) considered testing efforts and fault detection rates whereas Huang (2005a) considered testing efforts and change-point. Huang (2005a) argued that the fault detection rate does not remain constant over the intervals between fault occurnces, since fault detection rate strongly depends on the skill of test teams, program size, and software testability. Hence, it could change and thus in his study the faulty detection rate was changed during the software development process. Therefore, he incorporated both generalized logistic TEF and change-point parameters into software reliability modeling. On the other hand Kuo et al. (2001) also incorporated generalized logistic TEF. The fault detection rates were obtained from previous releases or other similar software projects and incorporated the testing activities into this new modeling approach. Several real data sets were demonstrated to show the applicability of the model and the related parametric decomposition method. The SRGM parameters were estimated by the methods of maximum likelihood estimation and least square estimation.

Huang *et al*. (2002) studied three optimal release allocation problems based on SRGMs with generalized logistic testing-effort function in modular software systems during testing phase. They proposed several useful optimization algorithms based on the Lagrange multiplier method. Furthermore, they introduced the testing–resource control problem and compared different resource allocation methods. They also demonstrated how these analytical approaches could be employed in the integration testing.

Lo *et al*. (2002) proposed an analytical approach for estimating the reliability of component-based software. This methodology assumes that the software components are heterogeneous and the transfers of control between components follow a discrete time Markov process. Besides, they also formulated and solved two resource allocation problems. Moreover, they demonstrated how these analytical approaches can be employed to measure the reliability of a software system including multiple-input/multiple-output systems and distributed software systems. According to them, their experiment showed that the proposed methods could solve the testing-effort allocation problems and improve the quality and reliability of a software system.

Xie and Yang (2003) studied the effect of imperfect debugging on a commonly used cost model. Also, the problem of determining the optimal testing level is studied with additional resources.

Kapur *et al*. (2004) proposed a Non-Homogeneous Poisson Process based SRGM with testing effort described by exponential and Rayleigh curves, to represent the growth curve of the fault removal process of the software, which contains two types of faults: mutually independent and mutually dependent. The model has validated on different data sets. Using the developed model, the testing resources were allocated optimally to modular software subject to different constraint. These optimization problems are mathematical programming problems which were solved using a dynamic programming approach. The SRGM parameters were estimated by the method of maximum likelihood estimation and least square estimation.

Berman and Cutler (2004) developed a framework for performing resource allocation (budget and time) during the test process of a software system. The framework allows the usage of different reliability models. They assumed that a software system has been specified, designed and coded, and that a test plan for testing the system is available. A model has been developed with the goal of finding the maximum reliability of the software system while satisfying the following constraints: total test cost cannot exceed a given budget and requirements regarding the number of test and repair periods, and minimum reliability of components must be satisfied. The model has been solved for a variety of different constraints and parameter values using the Solver Add-in of Microsoft Excel.

Huang (2005) reviewed a SRGM with generalized testing-effort function which could be used to describe the actual consumption of resources during the software development process. He also proposed a software cost model that could be used to formulate realistic total software cost projects and discuss the optimal release policy based on cost and reliability considering testing effort and efficiency. According, to Huang, the problem of how to decide when to stop testing and when to release software for use could specifically be addressed based on the proposed models and methods.

Huang and Lyu (2005) considered two kinds of software testing-resource allocation problems. The first problem was to minimize the number of remaining faults given a fixed amount of testing-effort and a reliability objective and the second problem was to minimize the amount of testing-effort given the number of remaining faults and a reliability objective. Several numerical examples on the optimal testing-resource allocations were done to show the applications and impacts of the proposed strategies during module testing. Moreover, an extensive sensitivity analysis was presented to investigate the effects of various principal parameters on the optimization problem of testing-resource allocation. The results helps in determining which parameter have the most significant influence, and the changes of optimal testing-effort expenditures affected by the variations of fault detection rate and expected initial faults.

Lo (2005) studied two optimal resource allocation problems in module software during testing phase: minimization of the remaining faults when a fixed amount of testing-effort is given and secondly minimization of the required amount of testing-effort when a specific reliability requirement is given. Several useful optimization algorithms based on Lagrange multiplier method are proposed. In addition, they presented the sensitivity analysis on these allocation problems in order to determine which of the parameters affects the system most. For validation and to show the effectiveness, a numerical example was evaluated.

Huang and Lo (2006) presented an optimal resource allocation problem in modular software system during testing phase. The main purpose was to minimize the cost of software development when the fixed amount of testing-effort and a desired reliability objective was given. An elaborated optimization algorithm based on the Lagrange multiplier method was proposed with numerical examples illustrated. Moreover, sensitivity analysis was also performed.

Bokhari and Ahmad (2006) and Quadri *et al.* (2006) developed a SRGM based on NHPP which incorporated the amount of testing-effort consumptions during the software testing phase. Bokhari and Ahmad (2006) incorporated Log-Logistic TEF whereas Quadri *et al.* (2006) incorporated generalized exponential TEF. They assumed that the fault detection rate to the amount of testing-effort spent during the testing phase is proportional to the current fault content. The software reliability measures such as the number of remaining faults, fault detection rate and the estimation methods of the model parameters were investigated through a numerical experiment on real data sets from a software project. The model was analyzed by comparing with other existing models, to show that the proposed SRGM had a fairly better fault prediction capability. Moreover, Bokhari and Ahmad (2006) discussed the optimal release policy based on reliability and cost criteria for software systems.

Pham (2007) proposed a software reliability model connecting the imperfect debugging and learning phenomenon by a common parameter among the two functions, called the imperfect-debugging fault-detection dependent-parameter model. He also aimed to study the fact if the model parameters of both the fault content rate function and fault detection rate function of the SRGMs that are considered to be independent of each other. The proposed model was illustrated by using software testing data from real applications for both the descriptive and predictive power by determining non-zero initial debugging process.

Ahmad *et al*. (2008) proposed a SRGM based on NHPP for software testing which incorporated the Exponentiated Weibull (EW) testing-effort function. Moreover, they discussed the optimal release policy for this model based on cost-reliability criteria. MLE and LSE methods were used to estimate the SRGM parameters.

Ahmad *et al*. (2008a) investigated Inflection S-Shaped SRGM based on NHPP which incorporated the Exponentiated Weibull (EW) testing-effort function. Furthermore, the model was also discussed under the imperfect debugging environment. In the paper, the model parameters and testing-effort function parameters were estimated by the maximum likelihood estimation and the least square estimation methods, respectively. The software reliability measures were investigated through numerical experiments on actual data from three software projects. Results were then compared with other existing models and thus they claimed that their proposed SRGM had a fairly better prediction capability.

CHAPTER 3: A Study of Testing-effort Dependent Inflection S-shaped Software reliability Growth Models with Imperfect Debugging

3.1 Introduction

Software is considered to have performed a successful operation, when it functions completely as expected, without any failure. However, computer software is created by human being and a high degree of certainty in reliability cannot be guaranteed (Musa *et al*. 1987). Software reliability is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment (Musa *et al*. 1987; Lyu, 1996) and is a key factor in software development process. Therefore, accurately modeling software reliability and predicting its possible trends are essential for determining product's overall reliability. Numerous software reliability growth models (SRGMs) have been developed during the last three decades and they have been applied successfully in practice to improve software reliability (Musa *et al*. 1987; Xie, 1991; Lyu, 1996; Pham, 2000; Huang 2005; Quadri *et al*. 2006; Ahmad *et al*. 2008).

In software testing, the key factor is the testing-effort, which can be represented as the number of executed test cases, the number of CPU hours or the calendar time, the amount of man-power, and some other suitable measures (Yamada and Osaki, 1985; Yamada *et al*. 1986, 1993). The measures used usually depend on the products. Non-Homogeneous Poisson process (NHPP) as the stochastic process has been widely used in SRGM. Several SRGMs based on NHPP which incorporates the testing–effort functions (TEF) have been proposed by many authors (Yamada *et al*. 1986; 1987; 1993; Yamada and Ohtera, 1990; Kapur *et al*. 1994; Kapur and Younes, 1996; Huang *et al*. 1997; 2007; Kuo *et al*. 2001; Huang and Kuo, 2002; Huang, 2005; 2005a; Bokhari and Ahmad, 2006; Quadri *et al*. 2006; 2008; 2008a; Ahmad *et al*. 2008; 2009). Most of these works on SRGMs modified the exponential NHPP growth model (Goel and Okumoto, 1979) and incorporated the concept of testing-effort into an NHPP model to describe the software fault detection phenomenon. However, the exponential NHPP growth model is sometimes insufficient and inaccurate to analyze real software failure data for reliability assessment. This is due to non-homogeneous time distribution of the testing-effort. When calendar-time based data are used, a basic assumption for the exponential model is that the testing-effort is homogeneous throughout the testing phase (Ohba, 1984).

This chapter first reviews the EW testing-effort functions and discusses how to incorporate the EW testing-effort function into inflection S-shaped NHPP growth models (Ohba, 1984; 1984a) to get a better description of the software fault detection phenomenon. The proposed framework is a generalization over the previous works on SRGMs with testing-efforts (Yamada *et al*. 1986; 1987; 1990; 1993; Kapur *et al*. 1994; Kapur and Younes, 1996; Quadri *et al*. 2006; 2008; Bokhari and Ahmad, 2007; Ahmad *et al*. 2008; 2009). The parameters of the model are estimated by Weighted Least Square Estimation (WLSE) and Maximum Likelihood Estimation (MLE) methods. The statistical methods of data analysis are presented and the experiments are performed based on real data sets and the results are compared with other existing models. The experimental results shows that the proposed SRGMs with EW testing-effort function can estimate the number of initial faults better than that of other models and that the EW testing-effort functions is suitable for incorporating into inflection S-shaped NHPP growth models. Further, the analysis of the proposed models under imperfect debugging environment is also discussed.

3.2 Review of Exponentiated Weibull TEF

During the software testing phase, much testing-effort is consumed. The consumed testing-effort indicates how the faults are detected effectively in the software and can be modeled by different distributions (Musa *et al*. 1987; Yamada *et al*. 1986; 1993; Kapur *et al*. 1999). Actually, the software reliability is highly related to the amount of testing-effort expenditures spent on detecting and correcting software faults. Recently, Bokhari and Ahmad (2007) and Ahmad *et al*. (2008, 2009) proposed EW testing-effort function to predict the behavior of failure and fault of a software product. They have shown that EW testing-effort function is very suitable and more flexible testing resource for assessing the reliability of software products.

The cumulative testing-effort expenditure consumed in (0, *t*] is depicted in the following:

$$W(t) = \alpha(1 - e^{-\beta\, t^{\delta}})^{\theta}, \alpha > 0, \beta > 0, \delta > 0, \theta > 0\,, \qquad (1)$$

and the current testing-effort consumed at testing time *t* is

$$w(t) = \frac{dW(t)}{dt} = \alpha.\beta.\delta.\theta.t^{\delta-1}.e^{-\beta\, t^{\delta}}(1 - e^{-\beta\, t^{\delta}})^{\theta-1}\;, \qquad (2)$$

where α is the total amount of testing-effort expenditures; β is the scale parameter, and δ and θ are shape parameters.

There are 5 special cases.

- Yamada exponential curve (Yamada *et al*. 1986): For θ = 1 and δ = 1, there is an exponential testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta\, t}), \alpha > 0, \beta > 0.$$

- Yamada Rayleigh curve (Yamada *et al*. 1986): For θ = 1 and δ = 2, there is a Rayleigh testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta\, t^{2}}), \alpha > 0, \beta > 0.$$

- Yamada Weibull curve (Yamada *et al*. 1993): For θ = 1, there is a Weibull TEF, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta\, t^{\delta}}), \alpha > 0, \beta > 0, \delta > 0\,.$$

- Generalized exponential curve (Quadri *et al*. 2006; 2008a): For δ = 1, there is a generalized exponential testing-effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t})^{\theta}, \alpha > 0, \beta > 0, \theta > 0.$$

- Burr Type X curve (Ahmad *et al.* 2009; Quadri *et al.* 2008): For $\delta = 2$, there is a Burr Type X testing effort function, and the cumulative testing-effort consumed in time $(0, t]$ is

$$W(t) = \alpha(1 - e^{-\beta t^2})^{\theta}, \alpha > 0, \beta > 0, \theta > 0.$$

3.3 Software Reliability Growth Models

Software reliability growth models (SRGMs) provides very useful information about how to improve reliability. Past decades several SRGMs have been developed (Musa *et al.* 1987; Xie, 1991; Lyu, 1996; Huang, 2005; Quadri *et al.* 2006; Ahmad *et al.* 2008, 2009), among which exponential growth model and inflection S-shaped growth model have been shown to be very useful in fitting software failure data. Many authors (Yamada *et al.* 1986, 1993; Musa *et al.* 1987; Huang et al. 1997, 2002; Kapur *et al.* 2004; Quadri *et al.* 2006; Ahmad *et al.* 2008) incorporated the concept of testing-effort into exponential type SRGM based on the NHPP to get a better description of the fault detection phenomenon. The following section shows how to incorporate the EW testing-effort function into inflection S-shaped NHPP growth models.

3.4 Inflection S-Shaped SRGM with EW Testing-Effort Function

Ohba (1984; 1984a) reported that the exponential growth model is sometimes insufficient and inaccurate to analyze real software failure data for reliability assessment. Hence, he raised the inflection S-shaped NHPP software reliability growth model. The model has been shown to be useful in fitting software failure data. Ohba proposed that the fault removal rate increases with time and assumed the presences of two types of faults in the software. Later, Kapur *et al.* (2004) and

Ahmad *et al*. (2008a) modified the inflection S-shaped model and incorporated the testing-effort in an NHPP model.

The extended inflection S-shaped SRGM with EW testing-effort function is formulated on the following assumptions (Ohba, 1984; 1984a; Yamada and Osaki, 1985; Yamada *et al*. 1986; 1993; Kapur *et al*. 1999; Kuo *et al*. 2001; Huang, 2005; Huang et al. 2007; Kapur *et al*. 2004; Bokhari and Ahmad, 2007; Ahmad *et al*. 2008; 2008a):

1. The software system is subject to failures at random times caused by faults remaining in the system.
2. Fault removal phenomenon in software testing is modeled by Non-Homogeneous Poisson Process (NHPP).
3. The mean number of faults detected in the time interval $(t, t+\Delta t]$ by the current testing-effort expenditures is proportional to the mean number of detectable faults in the software.
4. The proportionality increases linearly with each additional fault removal.
5. Testing-effort expenditures are described by the EW testing-effort function.
6. Each time a failure occurs, the fault caused the failure is immediately removed and no new faults are introduced.
7. Faults present in the software are of two types: mutually independent and mutually dependent.

The mutually independent faults lie on different execution paths, and mutually dependent faults lie on the same execution path (Kapur *et al*. 2004). Thus, the second type of faults is detectable if and only if the faults of the first type have been removed. According to these assumptions, if the fault detection rate with respect to current testing-effort expenditures is proportional to the number of detectable faults in the software and the proportionality increases linearly with each additional fault removal, the following differential equation is obtained:

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = \phi(t) \;\; a - m(t) \;\; , \tag{3}$$

where

$$\phi(t) = b\left[r + (1-r)\frac{m(t)}{a} \right],$$

$(r > 0)$ is the inflection rate and represents the proportion of independent faults present in the software, $m(t)$ is the mean value function of the expected number of faults detected in time $(0,t]$, $w(t)$ is the current testing-effort expenditure at time t , a is the expected number of faults in the system, and b is the fault detection rate per unit testing-effort at time t.

Solving (3) with the initial condition that, at $t = 0, W(t) = 0, m(t) = 0$, the following is obtained

$$m(t) = \frac{a\left[1 - e^{-bW(t)}\right]}{1 + \ (1-r)/r \ e^{-bW(t)}} . \qquad (4)$$

If the inflection rate r = 1, the above NHPP model becomes equivalent to the exponential growth model and is given as

$$m(t) = a\left[1 - e^{-bW(t)}\right].$$

Now the derivation of equation (4) from (3) will be shown:

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = \phi(t) \ \ a - m(t) \ \ ,$$

where $$\phi(t) = b\left[r + (1-r)\frac{m(t)}{a} \right],$$

To solve the above differential equation, method of separable variable is used.

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = \phi(t) \ \ a - m(t) \ \ ,$$

$$\frac{dm(t)}{\phi(t)[a-m(t)]}=w(t)dt,$$

$$\frac{dm}{b[r+(1-r)m/a][a-m]}=w(t)dt,$$

$$\frac{dm}{1/a[ar+(1-r)m][a-m]}=bw(t)dt,$$

$$\frac{dm}{a^2r-arm+a(1-r)m-(1-r)m^2}=\frac{b}{a}w(t)dt,$$

$$\frac{dm}{r\ \ a^2-am+a\ \ (1-r)/r\ \ m-\ \ (1-r)/r\ \ m^2}=\frac{b}{a}w(t)dt,$$

$$\frac{dm}{a(a-m)+\ \ (1-r)/r\ \ m(a-m)}=\frac{rb}{a}w(t)dt,$$

Factorising the denominator and apply Integral both sides, we get

$$\int\frac{dm}{[(1-r)/r]m+a\quad a-m}=\frac{rb}{a}\int w(t)dt, \qquad (5)$$

To evaluate the L.H.S of the above, the technique of partial fractions is applied.

$$\frac{dm}{[(1-r)/r]m+a\quad a-m}=\frac{A}{[(1-r)/r]m+a}+\frac{B}{a-m}, \qquad (6)$$

$$1=A(a-m)+B\quad (1-r)/r\ \ m+a\quad ,$$

let $m=a$ let $m=-ar/(1-r)$

$1=B[((1-r)/r)a+a]$ $1=A[a--ar/(1-r)]$

$1=Ba[1-r+r]/r$ $1=Aa[1-r+r]/(1-r)$

$1=Ba/r$ $1=Aa/(1-r)$

$B=r/a$ $A=(1-r)/a$

Hence substituting the values of *A* and *B* and putting the integral sign back into equation (6) gives

$$\int\frac{dm}{[(1-r)/r]m+a\quad a-m}=\int\left(\frac{1-r}{a\ \ [(1-r)/r]m+a}+\frac{r}{a\ \ a-m}\right)dm. \qquad (7)$$

Thus integrating L.H.S. of equation (5) is same as integrating R.H.S of equation (7). Rewriting gives

$$\int\left(\frac{1-r}{a\ \ [(1-r)/r]m+a}+\frac{r}{a\ \ a-m}\right)dm=\frac{rb}{a}\int w(t)dt,$$

Integrate in [0, *t*]

$$\frac{1-r}{a}\int_0^t\frac{1}{[(1-r)/r]m+a}dm+\frac{r}{a}\int_0^t\frac{1}{a-m}dm=\frac{rb}{a}\int_0^t w(t)dt,$$

Upon integration both sides give:

$$\left[\frac{(1-r)}{a}\times\frac{1}{(1-r)/r}\ln\ [(1-r)/r]m+a\ \ +\frac{r}{a}\times-\ln(a-m)\right]_0^t=\left[\frac{r}{a}bW(t)\right]_0^t\ ,$$

$$\left[\frac{r}{a}\ln\ [(1-r)/r]m+a\ \ -\frac{r}{a}\ln(a-m)\right]_0^t=\left[\frac{r}{a}bW(t)\right]_0^t\ ,$$

Simplifying the above and applying logarithm rule gives:

$$\frac{r}{a}\ln\frac{[(1-r)/r]m(t)+a}{(a-m(t))}=\frac{rb}{a}W(t)\ ,$$

$$\ln\frac{[(1-r)/r]m(t)+a}{a-m(t)}=bW(t)\ ,$$

Applying exponential both sides

$$\frac{[(1-r)/r]m(t)+a}{a-m(t)}=e^{bW(t)}\ ,$$

$$\frac{a-m(t)}{[(1-r)/r]m(t)+a}=e^{-bW(t)}\ ,$$

$$a-m(t)=e^{-bW(t)}\ \ [(1-r)/r]m(t)+a\ \ ,$$

Collecting like terms

$$a - ae^{-bW(t)} = [(1-r)/r]e^{-bW(t)}m(t) + m(t),$$

$$a(1-e^{-bW(t)}) = m(t)\ \ 1+[(1-r)/r]e^{-bW(t)}\ \ ,$$

$$m(t) = \frac{a(1-e^{-bW(t)})}{1+\ \ (1-r)/r\ \ e^{-bW(t)}}\ \ .$$

Thus, equation (4) is obtained.

The failure intensity at testing time t of the inflection S-shaped NHPP model with testing-effort is given by

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{a \cdot b \cdot w(t) \cdot e^{-bW(t)}}{r\left[1+((1-r)/r)e^{-bW(t)}\right]^2}\ . \tag{8}$$

The above is obtained by differentiating equation (4) using quotient rule

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ \ abw(t)e^{-bW(t)}\ \ -\ \ a(1-e^{-bW(t)}\ \ \ -bw(t)\ \ (1-r)/r\ \ e^{-bW(t)}}{1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ ^2},$$

$$\lambda(t) = \frac{abw(t)e^{-bW(t)} + abw(t)\ \ (1-r)/r\ \ e^{-2bW(t)} + abw(t)\ \ (1-r)/r\ \ e^{-bW(t)} - abw(t)\ \ (1-r)/r\ \ e^{-2bW(t)}}{1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ ^2},$$

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{abw(t)e^{-bW(t)}\ \ 1+(1-r)/r}{1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ ^2},$$

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{abw(t)e^{-bW(t)}\ \ (r+1-r)/r}{1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ ^2},$$

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{abw(t)e^{-bW(t)}}{r\ \ 1+\ \ (1-r)/r\ \ e^{-bW(t)}\ \ ^2}\ \ .$$

In addition, the expected number of faults to be detected eventually is

$$m(\infty)=\frac{a\left[1-e^{-b\alpha}\right]}{1+((1-r)/r)e^{-b\alpha}} \ .$$

3.5 Estimation of Model Parameters

The success of a software reliability growth model depends heavily on the quality of the failure data collected. The parameters of the SRGM are estimated based upon these data. In order to validate the proposed model and to compare its performance with other existing models, experiments on three real software failure data are performed. Maximum Likelihood estimation (MLE) and Weighted Least Square estimation (WLSE) techniques are used to estimate the model parameters (Musa *et al*. 1987; Musa, 1999; Lyu, 1996; Quadri *et al*., 2008; Ahmad *et al*. 2009).

3.5.1 Weighted Least Square Method

Using the method of Weighted Least Square estimation, the parameters α , β , δ , and θ in Exponentiated Weibull testing-effort function is estimated. These parameters are determined for n observed data pairs in the form (t_k, W_k); $(k=1,2,....,n;\quad 0<t_1<t_2<....<t_n)$, where W_k is the cumulative testing-effort consumed in time $(0,t_k]$. The estimators $\hat{\alpha}$, $\hat{\beta}$, $\hat{\delta}$ and $\hat{\theta}$, which contribute the model with a greater fitting, are obtained by minimizing equation (9) (Bokhari and Ahmad, 2007; Quadri *et al*. 2008; Ahmad *et al*. 2008; 2009):

$$\mathrm{S}(\alpha,\beta,\delta,\theta)=\sum_{k=1}^{n} l_k \;\; W_k - W(t_k)\;^2 = \sum_{k=1}^{n} l_k \left[\ln W_k - \ln\alpha - \theta\ln(1-e^{-\beta\, t_k^{\delta}})\right]^2 . \qquad (9)$$

where l_k are weights assigned to data points according to their proper amount of influence over the parameter estimates.

Differentiating $S\;\;\alpha,\beta,\delta,\theta$ with respect to $\alpha,\beta,\;\delta$, and θ, and equating the partial derivatives to zero, the following non-linear normal equations are obtained:

$$\frac{\partial S}{\partial\alpha}=2\sum_{k=1}^{n}l_k\;\;\ln W_k-\ln\alpha-\theta\ln\;\;1-e^{-\beta\cdot t_k^{\delta}}\;\;\;\cdot\left(-\frac{1}{\alpha}\right)=0\,,$$

$$\sum_{k=1}^{n}l_k\ln W_k-n\ln\alpha\sum_{k=1}^{n}l_k-\theta\sum_{k=1}^{n}l_k\ln\;\;1-e^{-\beta t_k^{\delta}}\;\;=0\,,$$

$$\sum_{k=1}^{n}l_k\ln W_k-\theta\sum_{k=1}^{n}l_k\ln(1-e^{-\beta t_k^{\delta}})=n\ln\alpha\sum_{k=1}^{n}l_k\;,$$

$$\frac{\sum_{k=1}^{n}l_k\ln W_k-\theta\sum_{k=1}^{n}l_k\ln(1-e^{-\beta t_k^{\delta}})}{n\sum_{k=1}^{n}l_k}=\ln\alpha\;,$$

$$e^{\left[\sum_{k=1}^{n}l_k\ln W_k-\theta\sum_{k=1}^{n}l_k\ln\left(1-e^{-\beta\cdot t_k^{\delta}}\right)\right]/n\sum_{k=1}^{n}l_k}=\hat{\alpha}\,,$$

Thus, the WLSE of α is given by

$$\hat{\alpha}=e^{\left[\sum_{k=1}^{n}l_k\ln W_k-\theta\sum_{k=1}^{n}l_k\ln\left(1-e^{-\beta\cdot t_k^{\delta}}\right)\right]/\sum_{k=1}^{n}l_k}\,. \tag{10}$$

The WLSE of θ can be obtain as

$$\frac{\partial S}{\partial\theta}=2\sum_{k=1}^{n}l_k\;\;\ln W_k-\ln\alpha-\theta\ln\;\;1-e^{-\beta\cdot t_k^{\delta}}\;\;\;\cdot\ln\;\;1-e^{-\beta\cdot t_k^{\delta}}\;\;=0\,,$$

$$=\sum_{k=1}^{n}l_k\ln W_k\ln\;\;1-e^{-\beta t_k^{\delta}}\;\;-\;\ln\alpha\sum_{k=1}^{n}l_k\ln\;\;1-e^{-\beta t_k^{m}}\;\;-\theta\sum_{k=1}^{n}l_k\left[\ln\;\;1-e^{-\beta t_k^{m}}\;\;\right]^2,$$

and hence

$$\hat{\theta} = \frac{\sum_{k=1}^{n} l_k \ln W_k \ln \; 1-e^{-\beta\cdot t_k^{\delta}} \; - \ln\alpha \cdot \sum_{k=1}^{n} l_k \ln \; 1-e^{-\beta\cdot t_k^{\delta}}}{\sum_{k=1}^{n} l_k \; \ln \; 1-e^{-\beta\cdot t_k^{\delta}} \; ^{2}} . \qquad (11)$$

Finally, the WLSE of β and δ can be obtain by substituting the above estimators into

$$\frac{\partial S}{\partial \beta} = 2\sum_{k=1}^{n} l_k \; \ln W_k - \ln\alpha - \theta \ln \; 1-e^{-\beta\cdot t_k^{\delta}} \; \cdot \left(-\frac{\theta \cdot t_k^{\delta} \cdot e^{-\beta\cdot t_k^{\delta}}}{1-e^{-\beta\cdot t_k^{\delta}}} \right) = 0 ,$$

$$= \sum_{k=1}^{n} l_k \frac{\left[\ln W_k - \ln\alpha - \theta \ln(1-e^{-\beta t_k^{\delta}}) \right] \; t_k^{\delta} \cdot e^{-\beta t_k^{\delta}}}{1-e^{-\beta t_k^{\delta}}} = 0 \; . \qquad (12)$$

and

$$\frac{\partial S}{\partial \delta} = 2\sum_{k=1}^{n} l_k \; \ln W_k - \ln\alpha - \theta \ln \; 1-e^{-\beta\cdot t_k^{\delta}} \; \cdot \left(-\frac{\theta\beta \cdot t_k^{\delta} \ln t_k \cdot e^{-\beta\cdot t_k^{\delta}}}{1-e^{-\beta\cdot t_k^{\delta}}} \right) = 0,$$

$$= \sum_{k=1}^{n} l_k \frac{\left[\ln W_k - \ln\alpha - \theta \ln(1-e^{-\beta t_k^{\delta}}) \right] \; t_k^{\delta} \cdot e^{-\beta t_k^{\delta}} \cdot \ln t_j}{1-e^{-\beta t_k^{\delta}}} = 0 . \qquad (13)$$

3.5.2 Maximum Likelihood Method

Once the estimates of α , β , δ , and θ are known, the parameters of the SRGMs can be estimated through Maximum Likelihood Estimation method. The estimators of a, b, and r are determined for the n observed data pairs in the form (t_k, y_k); $(k=1,2,...,n;\ 0 < t_1 < t_2 < ...... < t_n)$, where y_k is the cumulative number of software faults detected up to time t_k or $(0, t_k]$. Then the likelihood function for the unknown parameters a, b, and r in the NHPP model (4) is given by (Musa *et al*. 1987).

$$L'(a,b,r) \equiv P \;\; N(t_i) = y_i, \;\; i = 1,2,...,n$$

$$= \prod_{k=1}^{n} \frac{\left[m(t_k) - m(t_{k-1})\right]^{(y_k - y_{k-1})}}{(y_k - y_{k-1})!} . e^{-[m(t_k) - m(t_{k-1})]}, \tag{14}$$

where $t_0 \equiv 0$ and $y_0 \equiv 0$.

The following is obtained by taking logarithm on both sides in (14),

$$L = \ln L' = \sum_{k=1}^{n} (y_k - y_{k-1}) \ln \left[m(t_k) - m(t_{k-1})\right]$$

$$- \sum_{k=1}^{n} \left[m(t_k) - m(t_{k-1})\right] - \sum_{k=1}^{n} \ln \left[(y_k - y_{k-1})!\right].$$

Now

$$m(t_k) - m(t_{k-1}) = \frac{a(1-e^{-bW(t_k)})}{1+[(1-r)/r]e^{-bW(t_k)}} - \frac{a(1-e^{-bW(t_{k-1})})}{1+[(1-r)/r]e^{-bW(t_{k-1})}}.$$

Let $\lambda = \dfrac{1-r}{r}$,

$$= \frac{a(1-e^{-bW(t_k)})(1+\lambda e^{-bW(t_{k-1})}) - a(1-e^{-bW(t_{k-1})})(1+\lambda e^{-bW(t_k)})}{\left(1+\lambda e^{-bW(t_k)}\right)\left(1+\lambda e^{-bW(t_{k-1})}\right)},$$

Expanding and simplifying the numerator

$$= \frac{-ae^{-bW(t_k)}(1+\lambda) + ae^{-bW(t_{k-1})}(1+\lambda)}{\left(1+\lambda e^{-bW(t_k)}\right)\left(1+\lambda e^{-bW(t_{k-1})}\right)},$$

therefore,

$$m\left(t_k\right) - m\left(t_{k-1}\right) = \frac{a\left(1+\lambda\right)\left(e^{-bW\left(t_{k-1}\right)} - e^{-bW\left(t_k\right)}\right)}{\left(1+\lambda e^{-bW\left(t_k\right)}\right)\left(1+\lambda e^{-bW\left(t_{k-1}\right)}\right)}.$$

Hence,

$$\sum_{j=1}^{n} [m(t_k) - m(t_{k-1})] = [m(t_1) - m(t_0)] + [m(t_2) - m(t_1)] + [m(t_3) - m(t_2)]$$

$$+ .... + [m(t_{n-1}) - m(t_{n-2})] + [m(t_n) - m(t_{n-1})]$$

$$= - m(t_0) - m(t_n) \qquad (t_0 = 0, \ \ y_0 = 0)$$

$$\sum_{k=1}^{n}\Big[m\;\,t_k\;\;-m\;\,t_{k-1}\;\;\Big]=m\;\,t_n\;\;=\frac{a\Big[1-e^{-bW\;\,t_n\;\,}\Big]}{1+\lambda e^{-bW\;\,t_n\;\,}},$$

Thus,

$$L=\sum_{k=1}^{n}\;y_k-y_{k-1}\;\;\ln a+\sum_{k=1}^{n}\;y_k-y_{k-1}\;\;\ln\;1+\lambda\;\;+\sum_{k=1}^{n}\;y_k-y_{k-1}\;\;\ln\;e^{-bW\;\,t_{k-1}\;\,}-e^{-bW\;\,t_k}$$

$$-\sum_{k=1}^{n}\;y_k-y_{k-1}\;\;\ln\;1+\Big[\;1-r\;\;/r\Big]e^{-bW\;\,t_k}\;\;-\sum_{k=1}^{n}\;y_k-y_{k-1}\;\;\ln\;1+\Big[\;1-r\;\;/r\Big]e^{-bW\;\,t_{k-1}}$$

$$-\frac{a\Big[1-e^{-bW\;\,t_n\;\,}\Big]}{1+\Big[\;1-r\;\;/r\Big]e^{-bW\;\,t_n}}-\sum_{k=1}^{n}\ln\Big[\;y_k-y_{k-1}\;\;!\Big]\,. \tag{15}$$

The maximum likelihood estimates of SRGM parameters a, r and b can be obtained by solving the following equations.

$$\frac{\partial L}{\partial a}=\frac{\sum_{k=1}^{n}(y_k-y_{k-1})}{a}-\frac{a\Big[1-e^{-bW(t_n)}\Big]}{1+\;(1-r)/r\;\;e^{-bW(t_n)}}=0\,,$$

$$\frac{y_n}{a}=\frac{\Big[1-e^{-bW(t_n)}\Big]}{1+\;(1-r)/r\;\;e^{-bW(t_n)}}\,.$$

Let $\phi_n=e^{-b\,W(t_n)}$

$$\hat{a}=\frac{1+[(1-r)/r]\;\,\phi_n}{1-\phi_n}\quad. \tag{16}$$

$$\frac{\partial L}{\partial r}=\frac{\sum_{k=1}^{n}(y_k-y_{k-1})}{1+[(1-r)/r]}\times\left(-\frac{1}{r^2}\right)-\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{1+[(1-r)/r]e^{-bW(t_k)}}\times\left(-\frac{1}{r^2}e^{-bW(t_k)}\right)$$

$$-\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{1+[(1-r)/r]e^{-bW(t_{k-1})}}\times\left(-\frac{1}{r^2}e^{-bW(t_{k-1})}\right)-\frac{a\left[1-e^{-bW(t_n)}\right]}{\left[1+\ (1-r)/r\ e^{-bW(t_n)}\right]^2}\left(\frac{1}{r^2}e^{-bW(t_n)}\right).$$

Subtitute $\phi_k = e^{-bW(t_k)},\ k=1,2,\ldots n,$ and $\lambda=\dfrac{1-r}{r}$

$$\frac{\partial L}{\partial r}=\frac{-y_n}{r^2(1+\lambda)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_k}{r^2(1+\lambda\phi_k)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_{k-1}}{r^2(1+\lambda\phi_{k-1})}-\frac{a\phi_n\ 1-\phi_n}{r^2(1+\lambda\phi_n)^2}=0\,,$$

$$\frac{y_n}{r^2(1+\lambda)}+\frac{a\phi_n\ 1-\phi_n}{r^2(1+\lambda\phi_n)^2}=\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_k}{r^2(1+\lambda\phi_k)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_{k-1}}{r^2(1+\lambda\phi_{k-1})}=0. \qquad (17)$$

$$\frac{\partial L}{\partial b}=\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{[e^{-bW(t_{k-1})}-e^{-bW(t_k)}]}\times\ -W(t_{k-1})e^{-bW(t_{k-1})}+W(t_{k-1})e^{-bW(t_k)}$$

$$-\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{1+\lambda e^{-bW(t_k)}}\times -W(t_k)\lambda e^{-bW(t_k)}-\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{1+\lambda e^{-bW(t_k)}}\times -W(t_{k-1})\lambda e^{-bW(t_{k-1})}$$

$$-\frac{aW(t_n)e^{-bW(t_n)}(1+\lambda)}{1+\lambda e^{-bW(t_n)}\ ^{2}}=0\ ,$$

$$\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(W(t_k)\phi_k-W(t_{k-1})\phi_{k-1}}{[\phi_{k-1}-\phi_k]}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(\lambda W(t_k)\phi_k)}{1+\lambda\phi_k}$$
$$+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(\lambda W(t_{k-1})\phi_{k-1})}{1+\lambda\phi_{k-1}}=\frac{aW(t_n)\phi_n(1+\lambda)}{1+\lambda\phi_n\ ^{2}}\quad . \qquad (18)$$

The above can be solved by numerical methods to get the values of $\hat{a}$, $\hat{b}$ and $\hat{r}$.

Finally, if the sample size n of (t_k, y_k) is sufficiently large, then the maximum-likelihood estimates $\hat{a}$, $\hat{b}$ and $\hat{r}$ asymptotically follow a trivariate s-normal (TVN) distribution (Yamada *et al*. 1993; Yamada and Osaki, 1985; Kapur *et al*. 1999; 2004; Huang and Kuo, 2002). That is,

$$\begin{pmatrix}\hat{a}\\ \hat{b}\\ \hat{r}\end{pmatrix} \sim T\mathrm{VN}\left[\begin{pmatrix}\hat{a}\\ \hat{b}\\ \hat{r}\end{pmatrix} \quad , \quad V\right], \text{ as } n \to \infty . \tag{19}$$

The variance-covariance matrix V in the asymptotic properties of (19) is useful in qualifying the variability of the estimated parameters $\hat{a}$, $\hat{b}$ and $\hat{r}$, and is the inverse of the Fisher information matrix F, i.e., $V = F^{-1}$(Yamada *et al*. 1986; 1993; Yamada and Osaki, 1985), given by the expectation of the negative of the second partial derivative of L as

$$F = \begin{bmatrix} E\left[-\frac{\partial^2 L}{\partial a^2}\right] & E\left[-\frac{\partial^2 L}{\partial a \partial b}\right] & E\left[-\frac{\partial^2 L}{\partial a \partial r}\right] \\ E\left[-\frac{\partial^2 L}{\partial a \partial b}\right] & E\left[-\frac{\partial^2 L}{\partial b^2}\right] & E\left[-\frac{\partial^2 L}{\partial b \partial r}\right] \\ E\left[-\frac{\partial^2 L}{\partial a \partial r}\right] & E\left[-\frac{\partial^2 L}{\partial b \partial r}\right] & E\left[-\frac{\partial^2 L}{\partial r^2}\right] \end{bmatrix}. \tag{20}$$

The asymptotic variance- covariance matrix V of the maximum-likelihood estimates for $\hat{a}$, $\hat{b}$ and $\hat{r}$ is the inverse of the Fisher information matrix.

$$V = F^{-1} = \begin{bmatrix} Var(\hat{a}) & Cov(\hat{a},\hat{b}) & Cov(\hat{a},\hat{r}) \\ Cov(\hat{a},\hat{b}) & Var(\hat{b}) & Cov(\hat{b},\hat{r}) \\ Cov(\hat{a},\hat{r}) & Cov(\hat{b},\hat{r}) & Var(\hat{r}) \end{bmatrix}. \tag{21}$$

The $(1-\alpha)100\%$ confidence limits for a, b and r are obtained as (Yamada and Osaki, 1985):

$$\hat{a} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{a})} \le a \le \hat{a} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{a})}\,, \tag{22}$$

$$\hat{b} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{b})} \le b \le \hat{b} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{b})}\,, \tag{23}$$

and

$$\hat{r} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{r})} \le r \le \hat{r} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{r})}\,, \tag{24}$$

where $z_{\alpha/2}$ is the $(1-\alpha/2)$ quartile of the standard normal distribution.

3.6 Data Analysis and Model Comparison

3.6.1 Comparison Criteria

To check the performance of an SRGM with EW testing-effort function, the following four criteria are used:

1. The Accuracy of Estimation (AE) is defined (Musa *et al*. 1987; Yamada and Osaki, 1985; Huang and Kuo, 2002; Kuo *et al*. 2001) as

$$AE = \left| \frac{M_a - a}{M_a} \right|,$$

where M_a is the actual cumulative number of detected faults after the test, and a is the estimated number of initial faults. For practical purposes, M_a is obtained from software fault tracking after software testing.

2. The Mean of Squared Errors (MSE) (Long-term predictions) is defined (Lyu, 1996; Huang and Kuo, 2002: Kuo *et al*. 2001) as

$$MSE = \frac{1}{k}\sum_{i=1}^{k} \quad m(t_i) - m_i \quad ^{2},$$

Where $m(t_i)$ is the expected number of faults at time t_i estimated by a model, and m_i is the expected number of faults at time t_i. MSE gives a quantitative comparison for long-term predictions. A smaller MSE indicates a minimum fitting error and better performance (Huang *et al*. 1997; Kapur and Younes, 1996; Kapur *et al*. 1999).

3. The coefficient of multiple determination is defined (Musa *et al*. 1987; Musa, 1999) as

$$R^2 = \frac{S \quad \hat{\hat{\alpha}},0,1,1 \quad - S \quad \hat{\alpha},\hat{\beta},\hat{\delta},\hat{\theta}}{S \quad \hat{\hat{\alpha}},0,1,1},$$

where $\hat{\hat{\alpha}}$ is the LSE of α for the model with only a constant term, that is, $\beta = 0$, $\delta = 1$ and $\theta = 1$ in (6). It is given by $\ln \hat{\hat{\alpha}} = \frac{1}{n}\sum_{k=1}^{n} \ln W_k$. Therefore, R^2 measures the percentage of total variation about the mean accounted for by the fitted model and tells us how well a curve fits the data. It is frequently employed to compare models and assess which model provides the best fit to the data. The best model is the one which provides the higher R^2, that is, closer to 1 (Kumar *et al.* 2005; Ahmad *et al.* 2008). To investigate whether a significant trend exists in the estimated testing-effort, one could test the hypotheses $H_0 : \beta = 0, \delta = 1$ and $\theta = 1$, against $H_1 : \beta \neq 0$ or at least δ or $\theta \neq 1$ using F-test by merely forming the ratio

$$F = \frac{\left[S(\hat{\hat{\alpha}},0,1,1) - S(\hat{\alpha},\hat{\beta},\hat{\delta},\hat{\theta})\right]/3}{S(\hat{\hat{\alpha}},0,1,1)/(n-4)}.$$

If the value of F is greater than $F_\alpha(3, n-4)$, which is the α percentile of the F distribution with degrees of freedom 3 and $n-4$, it can be $(1-\alpha)100$ percent confident that H_0 should be rejected, that is, there is a significant trend in the testing-effort curve.

4. The Predictive Validity is defined (Musa *et al.* 1987; Musa, 1999) as the capability of the model to predict future failure behavior from present and past failure behavior. Assume that *q* failures are observed by the end of test time t_q. The failure data up to time t_e $(\leq t_q)$ is used to determine the parameters of *m*(*t*). Substituting the estimates of these parameters in the mean value function yields the estimate of the number of failures $\hat{m}(t_q)$ by t_q. The estimate is compared with the actually observed number *q*. This procedure is repeated for several of t_e. The ratio

$$\frac{\hat{m}(t_q) - q}{q}$$

is called the relative error. Values close to zero for relative error indicate more accurate prediction and hence a better model. The predictive validity can be visually checked by plotting the relative error for normalized test time t_e/t_q.

3.6.2 Model Comparison with Actual Data Set

Data Set 1:

Test Time (weeks)	**Cumulative execution time (CPU hours)**	**Cumulative faults**	**Test Time (weeks)**	**Cumulative execution time (CPU hours)**	**Cumulative faults**
1	2.45	15	11	26.23	233
2	4.90	44	12	27.76	255
3	6.86	66	13	30.93	276
4	7.84	103	14	34.77	298
5	9.52	105	15	38.61	304
6	12.89	110	16	40.91	311
7	17.10	146	17	42.67	320
8	20.47	175	18	44.66	325
9	21.43	179	19	47.65	328
10	23.35	206			

The first set of actual data is from the study by Ohba (1984). The system is PL/1 data base application software, consisting of approximately 1,317,000 lines of code. During 19 faults weeks of testing, 47.65 CPU hours were consumed and about 328 software faults were removed. Moreover, the total cumulative number of detected faults after a long time of testing was 358. Similarly, this testing effort data are applied to estimate the parameters of the EW testing-effort function by using the WSLE method. The estimated parameters α, β, δ and θ are:

$$\hat{\alpha} = 114.576, \qquad \hat{\beta} = 0.0000332,$$

$$\hat{\delta} = 2.784, \qquad \hat{\theta} = 0.40067.$$

The other parameters of the SRGM in (4) can be solved numerically by the MLE method. The estimated values of the parameters are:

$$\hat{a} = 388.485, \qquad \hat{b} = 0.06065, \qquad \hat{r} = 0.3851.$$

Figure 1, shows the fitting of the EW testing-effort by using above estimates. The fitted curves and the actual software data are shown by solid and dotted lines, respectively.

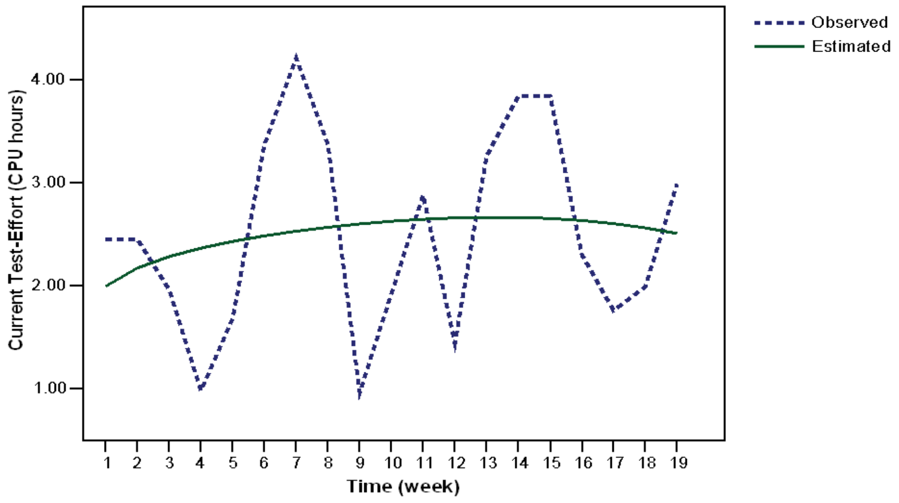

Figure 1: Observed/estimated current testing-effort function vs. time

Also, Figure 2 fits the estimated cumulative number of failures. The R^2 also known as the coefficient of determination, depicts how well a curve fits the data. A fit is more reliable when the value is closer to 1. The R^2 value for proposed EW testing-effort is 0.99578. Also, the calculated value $F(=4.979)$ is greater than $F_{0.05}(3,15)$. Therefore, it can be concluded that the EW testing-effort is suitable for modeling the proposed SRGM.

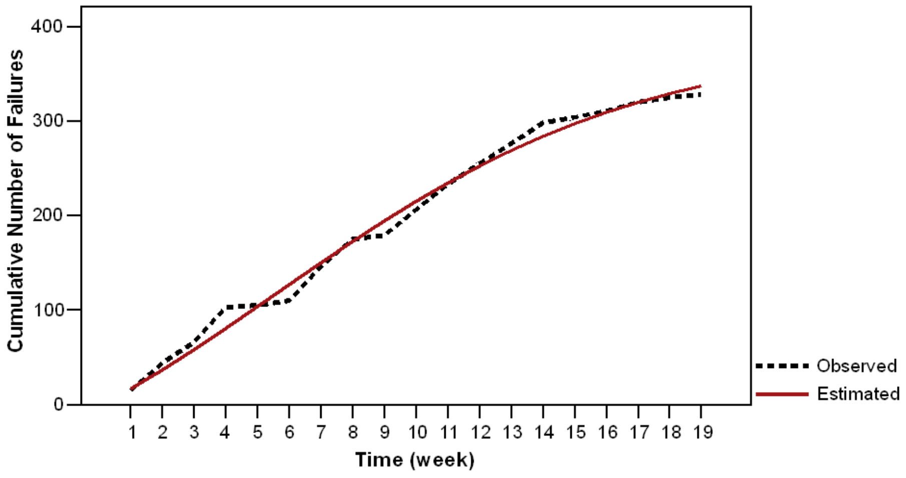

Figure 2: Observed/estimated cumulative number of failures vs. time

The comparisons of proposed model with different SRGMs are listed in Table I. It is observed that the proposed SRGM has smallest MSE. Kolmogorov Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

Table I: Comparison results of different SRGMs for Data Set 1

Model	***a***	***r***	***b***	**AE (%)**	**MSE**
Proposed model	388.49	0.381	0.06061	8.36	87.36
Ahmad EW model	565.64		0.01964	57.98	113.10
Huang Logistic model	394.08		0.04272	10.06	118.59
Yamada Rayleigh model	459.08		0.02734	28.23	268.42
Yamada Weibull model	565.35		0.01965	57.91	122.09
Yamada delayed S-shaped model	374.05		0.19765	4.48	168.67
Delayed S-Shaped model with Logistic TEF (Huang)	346.55		0.0936	3.20	147.61
Inflection S-shaped model	389.10	0.2	0.09355	8.69	133.53
G-O model	760.0		0.03227	112.29	139.815

Following the work of Musa *et al*. (1987), the relative error in prediction for this data set is computed and the results are plotted in Figure 3. Therefore, Figures 1-3 and Table I, reveal that the proposed model has better performance than the other models. This model fits the observed data better, and predicts the future behavior well.

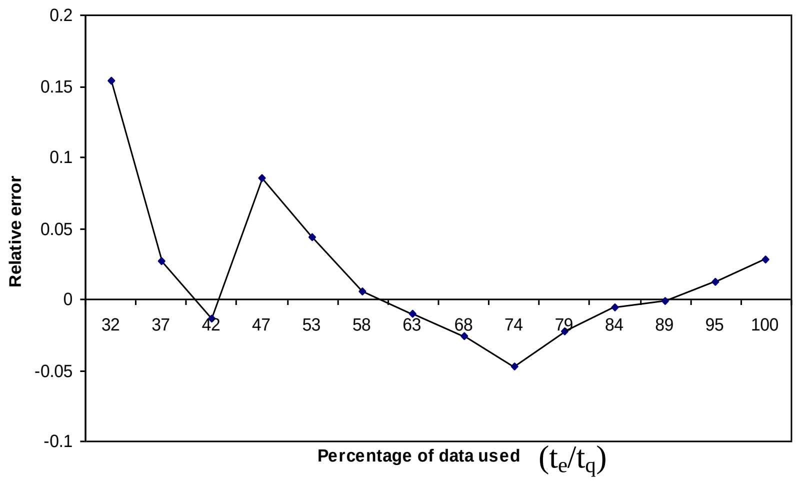

Figure 3: Predictive Relative Error Curve

Data Set 2:

Test Time (weeks)	**Cumulative execution time (CPU hours)**	**Cumulative faults**	**Test Time (weeks)**	**Cumulative execution time (CPU hours)**	**Cumulative faults**
1	519	16.00	11	6539	81.00
2	968	24.00	12	7083	86.00
3	1430	27.00	13	7487	90.00
4	1893	33.00	14	7846	93.00
5	2490	41.00	15	8205	96.00
6	3058	49.00	16	8564	98.00
7	3625	54.00	17	8923	99.00
8	4422	58.00	18	9282	100.00
9	5218	69.00	19	9641	100.00
10	5823	75.00	20	10000	100.00

The second set of actual data relates to the release of Tandem Computer Project cited in Wood (1996) from a subset of products for four separate software releases. In this research only Release 1 is used for illustrations. There were 10000 CPU hours consumed, over the 20 weeks of testing and 100 software faults were removed. The parameters α, β, δ and θ of the EW testing-effort function are estimated by using the WLSE method and are as follows:

$$\hat{\alpha} = 10710.073, \qquad \hat{\beta} = 0.002105,$$
$$\hat{\delta} = 2.283, \qquad \hat{\theta} = 0.56535.$$

The estimated parameters of the SRGM in (4) using MLE are:

$$\hat{a} = 198.96958567, \qquad \hat{b} = 9.4670 \times 10^{-6}, \qquad \hat{r} = 10.917276873.$$

The comparisons between the observed failure data and the estimated EW testing-effort data are illustrated graphically in Figure 4. Here, the fitted curves are shown as a solid line and the dotted line represents actual software data.

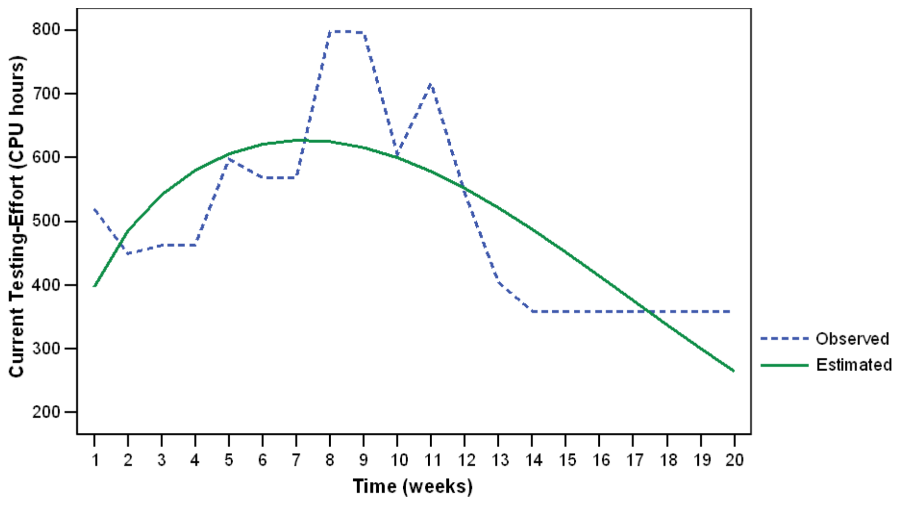

Figure 4: Observed/estimated current testing effort vs. time

The R^2 value for proposed EW testing-effort is 0.99818, which is very close to one. Moreover, the value of MSE is 12.59, which is small compared to other SRGMs as listed in the Table II. Also the fitted testing-effort curve is significant since the calculated value $F\,(=5.3236)$ is greater than $F_{0.05}(3,16)$.

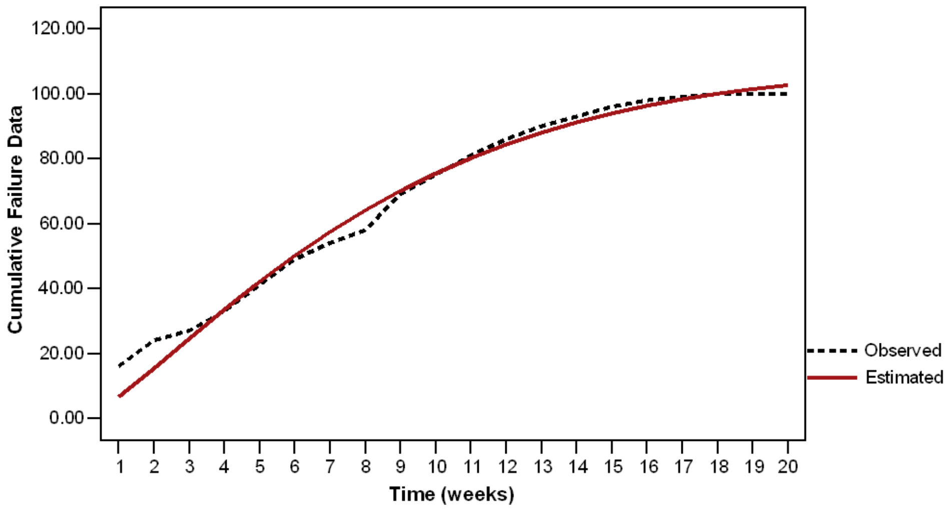

Figure 5: Observed/estimated cumulative number of failures vs. time

Table II lists the comparisons of proposed model with different SRGMs which reveal that the proposed model has better performance. It can therefore be observed that the EW testing-effort function is suitable for modeling the proposed SRGM of this data set. Kolmogorov Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

Table II: Comparison results of different SRGMs for Data Set 2

Model	***a***	***r***	***b***	**MSE**
Proposed model	198.97	10.917	9.467E-06	12.59
Ahmad EW	135.81		1.42E-04	13.31
Huang Logistic model	107.66		2.66E-04	22.76
Yamada Rayleigh model	110.60		2.26E-04	39.69
Yamada Weibull model	135.74		1.42E-04	18.43
Yamada delayed S-shaped model	102.26		0.345	94.99
Delayed S-shaped model with Logistic TEF	101.86		6.35E-04	92.66
Delayed S-shaped model with Rayleigh TEF model	102.14		0.000578	107.97
Inflection S-Shaped model	147.74	1.573	0.172	12.99
G-O model	137.072		0.0515445	25.33

Finally, the relative error in prediction of proposed model for this data set is computed. Figure 6 shows the relative error plotted against the percentage of data used (that is, t_e/t_q). Altogether, from Figures 4-6 and Table II, it is seen that the proposed model has better performance and predicts the future behavior well.

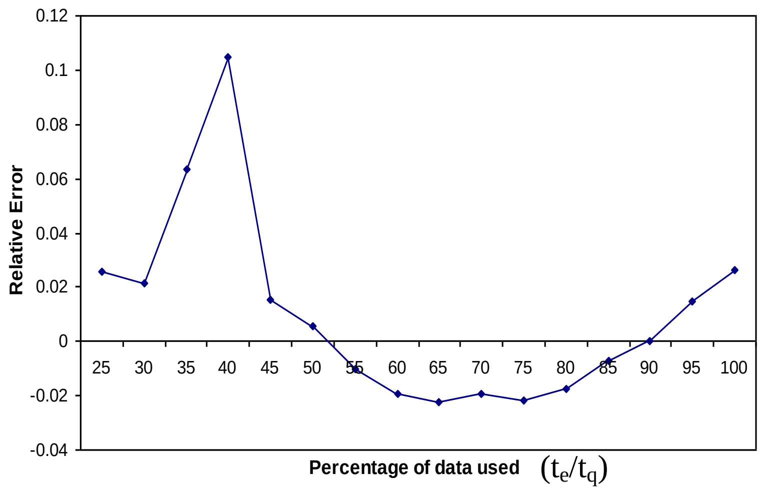

Figure 6: Predictive Relative Error Curve

Data Set 3:

Test Time (weeks)	Cumulative execution time (CPU hours)	Cumulative faults	Test Time (weeks)	Cumulative execution time (CPU hours)	Cumulative faults
1	0.00917	2	12	2.11017	44
2	0.01917	2	13	3.05017	55
3	0.02217	2	14	4.39017	69
4	0.04517	3	15	7.71017	87
5	0.08617	4	16	11.27017	99
6	0.09017	6	17	13.93017	111
7	0.11517	7	18	17.70017	126
8	0.41717	16	19	21.10017	132
9	1.39017	29	20	23.50017	135
10	1.41017	31	21	25.30017	136
11	1.86017	42			

The third set of actual data in this research is the System T1 data of the Rome Air Development Center (RADC) projects and cited from Musa *et al*. (1987), Musa (1999). The number of object instructions in System T1 which is a real-time command and control application is 21,700. The software was tested by nine testers over the period of 21 weeks. Through the testing phase, about 25.3 CPU hours were consumed and 136 software faults are removed. The number of faults removed after 3.5 years of test was reported to be 188 (Huang, 2005). Similarly, this testing effort data is applied to estimate the α, β, δ and θ of the EW testing-effort function by using the WLSE method. The estimated parameters of the EW testing-effort are:

$$\hat{\alpha} = 27.573, \qquad \hat{\beta} = 1.81\text{E-}07,$$
$$\hat{\delta} = 5.431, \qquad \hat{\theta} = 1.236 .$$

The estimated values of the parameters $a, b,$ and r in (4) using MLE are:

$$\hat{a} = 161.686, \qquad \hat{b} = 0.0036, \qquad \hat{r} = 47.275 .$$

Figures 7 and 8 shows the fitting of the EW testing-effort and cumulative number of failures with this data set respectively. Solid and dotted lines respectively show the fitted curves and the actual software data. The R^2 value for proposed EW testing-effort is 0.9974. The value of MSE for the proposed SRGM is small compared to other SRGMs as listed in the Table III. Therefore, it can be said that the proposed curve is suitable for modeling the software reliability. Also, the calculated value $F(=5.6517)$ is greater than $F_{0.05}(3,17)$ and $F_{0.01}(3,17)$, which concludes that the fitted testing-effort curve is highly significance for this data set.

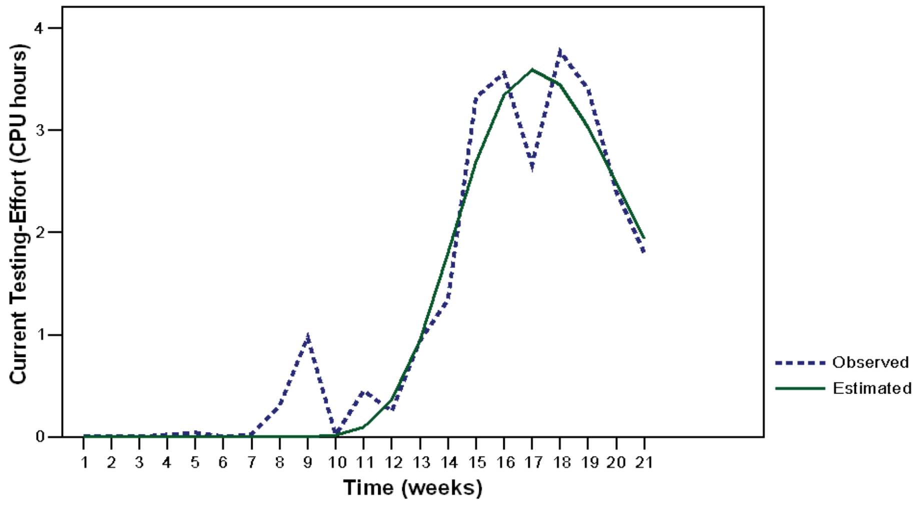

Figure 7: Observed/estimated current testing-effort function vs. time

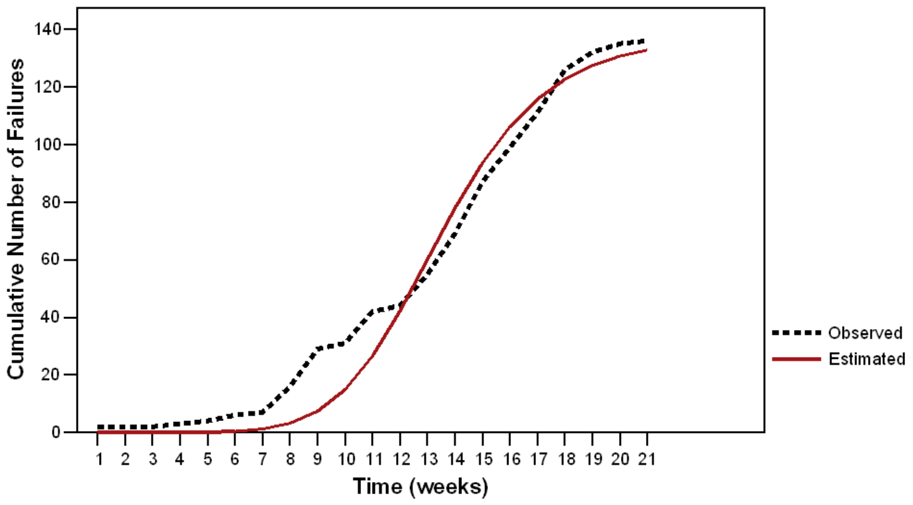

Figure 8: Observed/estimated cumulative number of failures vs. time

Table III lists the comparisons of proposed model with different SRGMs which reveal that the proposed model has better performance. Kolmogorov Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

Table III. Comparison results of different SRGMs for Data Set 3

Model	***a***	***r***	***b***	**AE (%)**	**MSE**
Proposed model	161.69	47.275	0.0036	13.99	63.95
Ahmad EW model	133.87		0.1546	28.79	78.55
Huang Logistic model	138.026		0.1451	26.58	64.41
Yamada Rayleigh model	866.94		0.0096	25.11	89.24
Yamada Weibull model	133.71		0.155	28.88	81.51
Yamada delayed S-Shaped model	237.19		0.0963	26.16	245.24
Delayed S-Shaped model with Logistic TEF (Huang)	124.11		0.411	33.98	180.02
Inflection S-shaped model	159.11		0.0765	15.36	118.3
G-O model	142.32		0.1246	24.29	2438.3

Lastly, the relative error in prediction of proposed model for this data set is calculated and shown graphically in Figure 9. The relative error is plotted against the percentage of data used (that is, t_e / t_q). Consequently, from the Figures 7 to 9 and Table III, it can be concluded that the proposed model gets reasonable prediction in estimating the number of software faults and fits the observed data better than the others.

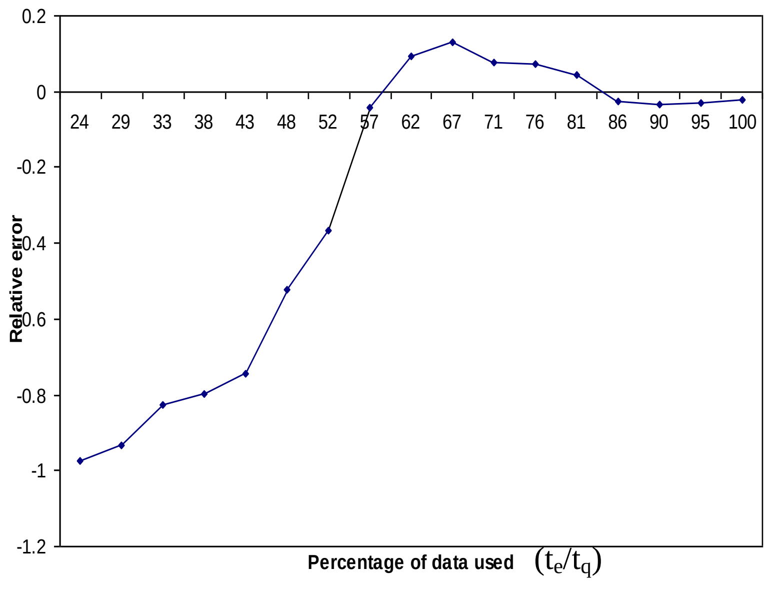

Figure 9: Predictive Relative Error Curve

3.7 Imperfect Debugging Model

Many researches have been conducted over the past decades to address the problem of imperfect debugging (Lo and Huang, 2004; Huang *et al*. 2007; Pham, 2007; Pham *et al*. 1999; Yamada *et al*. 1992; Zhang *et al*. 2003; Xie and Yang, 2003). In fact, to propose a new SRGM, the imperfect debugging problem has to be taken into consideration. In this section, the imperfect debugging of proposed SRGM is discussed. The perfect debugging assumption 6 given in section 3.4 can be modified as:

When a fault *is detected and removed, new* faults *may be generated. It is also assumed that if detected* faults *are removed, then there is a probability to introduce new* faults *with a constant rate* γ.

Based on the above modified assumption and the assumptions (1-5) mentioned in section 3.4, the SRGM with the EW testing-effort function within an imperfect debugging environment can be demonstrated in detail. Let $n(t)$ be the number of faults to be eventually detected plus the number of new faults introduced to the system by time t, the following system of differential equations is obtained:

$$\frac{dm(t)}{dt}\times\frac{1}{w(t)}=\phi(t)\ \ n(t)-m(t)\ \ , \tag{25}$$

$$\frac{dn(t)}{dt}=\gamma\frac{dm(t)}{dt}, \tag{26}$$

where $\phi(t)=b\left[r+(1-r)\dfrac{m(t)}{n(t)}\right]$ and $n(0)=a$.

Solving the above system under the boundary conditions $m(0)=0$ and $W(0)=0$, the following mean value function of inflection S-shaped model with EW testing-effort under imperfect debugging is obtained.

$$m_{debug}(t)=\frac{a\left[1-e^{-b(1-r\gamma)W(t)}\right]}{1-\gamma\ +\ (1-r)/r\ \ e^{-b(1-r\gamma)W(t)}}\ . \tag{27}$$

The derivation of equation (27) is as follows:

$$\frac{dm(t)}{dt}\times\frac{1}{w(t)}=\phi(t)\ \ n(t)-m(t)\ \ ,$$

$$\frac{dm(t)}{\phi(t)[n(t)-m(t)]}=w(t)dt,$$

$$\frac{dm}{b[r+(1-r)m/a][a+\gamma m-m]}=w(t)dt,$$

$$\frac{dm}{r/a[a+[(1-r)/r]m][a-(1-\gamma)m]}=bw(t)dt,$$

$$\frac{dm}{[a+[(1-r)/r]m][a-(1-\gamma)m]}=\frac{rb}{a}w(t)dt,$$

Apply Integral both sides

$$\int\frac{dm}{a+[(1-r)/r]m\quad a-(1-\gamma)m}=\frac{rb}{a}\int w(t)dt, \tag{28}$$

To evaluate the L.H.S of the above, the technique of partial fractions is applied.

$$\frac{dm}{a+[(1-r)/r]m\quad a-(1-\gamma)m}=\frac{A}{a+[(1-r)/r]m}+\frac{B}{a-(1-\gamma)m}\ \ ,$$

$$1=A\ \ a-(1-\gamma)m\ \ +B\ \ a+[(1-r)/r]m$$

let $m=a/(1-\gamma)$ let $m=-ar/(1-r)$

$1=B[a+((1-r)/r)a/(1-\gamma)]$ $1=A[a+ar(1-\gamma)/(1-r)]$

$1=Ba[r(1-\gamma)+a(1-r)]/r(1-\gamma)$ $1=Aa[1-r+r(1-\gamma)]/(1-r)$

$1=Ba\ \ (1-r\gamma)/r(1-\gamma)$ $1=Aa\ \ (1-r\gamma)/(1-r)$

$B=r(1-\gamma)/a(1-r\gamma)$ $A=(1-r)/a(1-r\gamma)$

Hence substituting the values of *A* and *B* and taking integrals the above gives

$$\int\frac{dm}{a+[(1-r)/r]m\quad a-(1-\gamma)m}=\int\left(\frac{1-r}{a(1-r\gamma)\ \ a+[(1-r)/r]m}+\frac{r(1-\gamma)}{a(1-r\gamma)\ \ a-(1-\gamma)m}\right)dm$$

Thus integrating L.H.S. of equation (28) is same as integrating R.H.S of the above equation. Rewriting gives

$$\int\left(\frac{1-r}{a(1-r\gamma)\ \ a+[(1-r)/r]m}+\frac{r(1-\gamma)}{a(1-r\gamma)\ \ a-(1-\gamma)m}\right)dm=\frac{rb}{a}\int w(t)dt,$$

Integrate in [0, t]

$$\frac{1-r}{a(1-r\gamma)}\int_0^t\frac{1}{a+[(1-r)/r]m}dm+\frac{r(1-\gamma)}{a(1-r\gamma)}\int_0^t\frac{1}{a-(1-\gamma)m}dm=\frac{rb}{a}\int_0^t w(t)dt,$$

Upon integration both sides give:

$$\left[\frac{(1-r)}{a(1-r\gamma)}\times\frac{1}{(1-r)/r}\ln\ \ a+[(1-r)/r]m\ \ +\frac{r(1-\gamma)}{a(1-r\gamma)}\times-\frac{1}{(1-\gamma)}\ln(a-(1-\gamma)m)\right]_0^t=\left[\frac{r}{a}bW(t)\right]_0^t,$$

$$\left[\frac{r}{a(1-r\gamma)}\ln\ \ a+[(1-r)/r]m\ \ -\frac{r}{a(1-r\gamma)}\ln(a-(1-\gamma)m)\right]_0^t=\left[\frac{r}{a}bW(t)\right]_0^t,$$

At $t=0,\ m(t)=0,\ W(t)=0,$ and with some simplifications, the following is obtained:

$$\frac{r}{a(1-r\gamma)}\ln\frac{a+[(1-r)/r]m(t)}{a-(1-\gamma)m(t)}=\frac{rb}{a}W(t)\,,$$

$$\ln\frac{a+[(1-r)/r]m(t)}{a-(1-\gamma)m(t)}=b(1-r\gamma)W(t)\,,$$

Applying exponential both sides

$$\frac{a+[(1-r)/r]m(t)}{a-(1-\gamma)m(t)}=e^{b(1-r\gamma)W(t)},$$

$$\frac{a-(1-\gamma)m(t)}{a+[(1-r)/r]m(t)} = e^{-b(1-r\gamma)W(t)},$$

$$a-(1-\gamma)m(t) = ae^{-b(1-r\gamma)W(t)} + [(1-r)/r]e^{-b(1-r\gamma)W(t)}m(t),$$

Collecting like terms

$$a - ae^{-b(1-r\gamma)W(t)} = (1-\gamma)m(t) + [(1-r)/r]e^{-b(1-r\gamma)W(t)}m(t),$$

Divide both sides by $(1-\beta)$

$$\frac{a}{(1-\gamma)} - \frac{a}{(1-\gamma)}e^{-b(1-r\gamma)W(t)} = m(t) + \frac{(1-r)}{r(1-\gamma)}e^{-b(1-r\gamma)W(t)}m(t),$$

$$\frac{a}{(1-\gamma)}(1-e^{-b(1-r\gamma)W(t)}) = m(t)\left(1+\frac{(1-r)}{r(1-\gamma)}e^{-b(1-r\gamma)W(t)}\right),$$

$$m_{debug}(t) = \frac{\frac{a}{(1-\lambda)}(1-e^{-b(1-r\lambda)W(t)})}{1+\ (1-r)/r(1-\lambda)\ \ e^{-b(1-r\lambda)W(t)}},$$

$$m_{debug}(t) = \frac{a(1-e^{-b(1-r\lambda)W(t)})}{(1-\lambda)+\ (1-r)/r\ \ e^{-b(1-r\lambda)W(t)}}.$$

Also have

$$n(t) = \frac{a\left[1+\ \ (1-r)/r\ -\lambda\ \ e^{-b(1-r\lambda)W(t)}\right]}{1-\lambda\ +\ (1-r)/r\ \ e^{-b(1-r\lambda)W(t)}}. \tag{29}$$

Derivation of equation (29):

$$n(t) = a + \lambda m(t),$$

$$= a + \lambda\frac{a(1-e^{-b(1-r\lambda)W(t)})}{(1-\lambda)+\ (1-r)/r\ \ e^{-b(1-r\lambda)W(t)}},$$

Let $z = b(1-r\lambda)W(t)$

$$= \frac{a(1-\lambda)+a\;\;(1-r)/r\;\;e^{-z}+\lambda a(1-e^{-z})}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-z}} \quad ,$$

$$= \frac{1}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-z}}\;\;a(1-\lambda)+a\;\;(1-r)/r\;\;e^{-z}+a\lambda(1-e^{-z})\;\;,$$

$$= \frac{1}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-z}}\;\;a+a\;\;(1-r)/r\;\;e^{-z}+a\lambda e^{-z}\;\;,$$

$$= \frac{a\;\;1+\;\;(1-r)/r\;\;e^{-z}+\lambda e^{-z}}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-z}},$$

$$= \frac{a\;\;1+\;\;[(1-r)/r]-\lambda\;\;e^{-z}}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-z}},$$

Therefore,

$$n(t) = \frac{a\;\;1+\;\;[(1-r)/r]-\lambda\;\;e^{-b(1-r\lambda)W(t)}}{(1-\lambda)+\;\;(1-r)/r\;\;e^{-b(1-r\lambda)W(t)}} \quad .$$

Thus, the failure intensity function $\lambda(t)$ for imperfect debugging model is obtained by differentiating $m_{debug}(t)$ with respect to *t*. Quotient rule is applied:

$$\lambda(t) = \frac{dm_{debug}(t)}{dt} = \frac{(1-\lambda)+ve^{-zW(t)}\;\;\times azw(t)e^{-zW(t)} - a(1-e^{-zW(t)})\times\;\;-zw(t)ve^{-zW(t)}}{(1-\lambda)+ve^{-zW(t)}\;\;^{2}}$$

,

where $z = b(1-r\lambda)$ and $v = (1-r)/r$

$$= \frac{az(1-\lambda)w(t)e^{-zW(t)}+azvw(t)e^{-2zW(t)}+azvw(t)e^{-zW(t)}-azw(t)ve^{-2zW(t)}}{(1-\lambda)+ve^{-zW(t)}\;\;^{2}},$$

$$= \frac{azw(t)e^{-zW(t)}\;\;(1-\lambda)+v}{(1-\lambda)+ve^{-zW(t)}\;\;^{2}},$$

$$=\frac{ab(1-r\gamma)w(t)e^{-b(1-r\gamma)W(t)}\ \ (1-\gamma)+(1-r)/r}{(1-\gamma)+\ \ (1-r)/r\ \ e^{-b(1-r\gamma)W(t)}\ \ ^{2}},$$

$$=\frac{ab(1-r\gamma)(1-r\gamma)w(t)e^{-b(1-r\gamma)W(t)}}{r\ \ (1-\gamma)+\ \ (1-r)/r\ \ e^{-b(1-r\gamma)W(t)}\ \ ^{2}},$$

$$=\frac{ab(1-r\gamma)(1-r\gamma)w(t)e^{-b(1-r\gamma)W(t)}}{r\ \ (1-\gamma)+\ \ (1-r)/r\ \ e^{-b(1-r\gamma)W(t)}\ \ ^{2}},$$

Thus $\lambda(t)$ is given by

$$\lambda(t)=\frac{ab\ \ 1-r\gamma\ \ ^{2}\ w(t)e^{-b(1-r\gamma)W(t)}}{r\left[(1-\gamma)+\ \ (1-r)/r\ \ e^{-b(1-r\gamma)W(t)}\right]^{2}}\quad . \tag{30}$$

The expected number of remaining faults after testing time t is

$$m_{remaining}(t)=n(t)-m(t)=\frac{a(1-r\gamma)e^{-b(1-r\gamma)W(t)}}{r(1-\gamma)+(1-r)e^{-b(1-r\gamma)W(t)}}\quad . \tag{31}$$

3.7.1 Numerical Example

To discuss the issue of imperfect debugging for proposed SRGM, Data Set 1 is taken into account. In order to validate the proposed SRGM under imperfect debugging, MSE is selected as the evaluation criterion. The parameters $a,b,r,$ and γ in (27) can be solved by the method of MLE. Table IV shows the estimated parameters of the proposed SRGM and some selected models for comparison under imperfect debugging. It also gives the results of MSE. It is observed that the value of MSE of the proposed SRGM with EW testing-effort function is the lowest among all the models considered. Moreover, the estimated values γ of all the models are close to but not equal to zero, thus the fault removal phenomenon may not be pure perfect debugging process. A fitted curve of the estimated cumulative number of failures with the actual software data and the RE curve for the proposed SRGM with EW testing-effort function under imperfect debugging is illustrated by Figure 10 and 11 respectively.

Table IV. Comparison results of different SRGMs for Data Set 1 under Imperfect Debugging

Models	***a***	***r***	***b***	γ	**AE (%)**	**MSE**
Proposed model	369.46	0.39	0.0618	0.049	3.20	87.36
Ahmad EW	453.53		0.0245	0.198	26.68	113.20
Huang Logistic model	391.62		0.0420	0.012	9.39	114.09
Yamada Rayleigh model	399.02		0.0316	0.013	11.46	268.55
Yamada Weibull model	10.06		1.11	0.982	97.19	116.64
Yamada delayed S-Shaped	19.87		1.68	0.962	94.44	114.71
Delayed S-Shaped model with Logistic TEF (Huang)	335.39		0.124	0.012	6.32	634.60
Inflection S-Shaped model	256.37	0.34	0.201	0.330	28.39	98.21
G-O model	530.61		0.0463	0.287	48.22	222.09

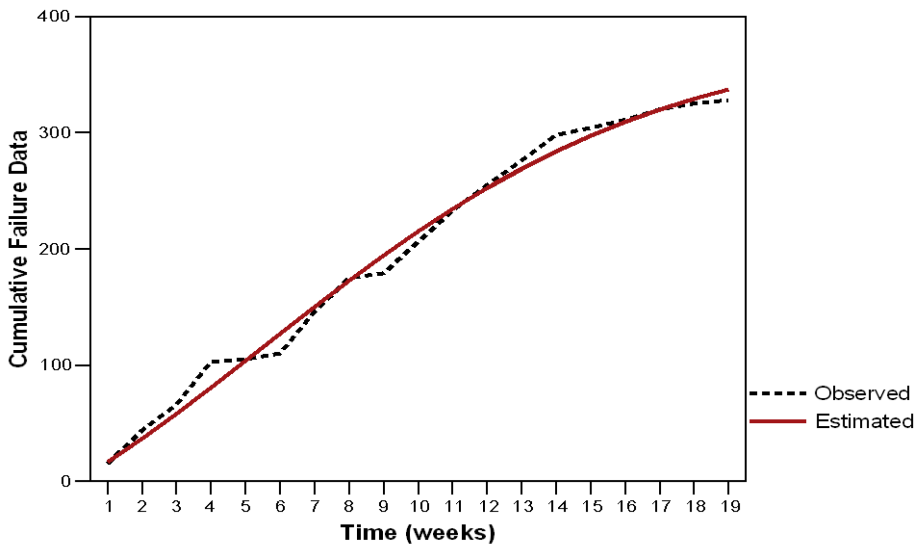

Figure 10: Observed/estimated cumulative number of failures vs. time

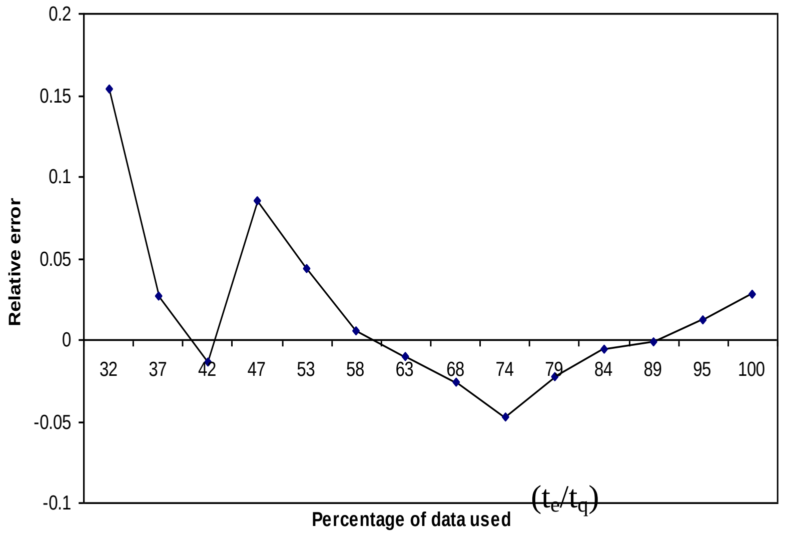

Figure 11: Predictive Relative Error Curve

CHAPTER 4: Analysis of an Inflection S-Shaped Software Reliability Model Considering Log-Logistic Testing Effort Function with Imperfect Debugging.

4.1 Introduction

The size and complexity of computer systems has grown significantly during the past decades. Before, the focus was only on the design and reliability of the hardware. But, now increase in the demand of software has led to the study of the high quality reliable software development. Reliability is the most important aspect since it measures software failures during the process of software development. Software embedded systems are used in safety critical application such as medicines, transportations and nuclear energy. Thus it leads to great demand for high quality and reliable software products. Many researches have been conducted over the past decades (Pham, 2000; Lyu, 1996; Musa *et al*. 1987) to study the software reliability. A common approach for measuring software reliability is by using an analytical model whose parameters are generally estimated from available data on software failures (Lyu, 1996; Musa *et al*. 1987).

A software reliability growth model (SRGM) is developed to evaluate software reliability (Lyu, 1996; Xie, 1991; Musa *et al.* 1987). Most of these models are developed for the analysis of software failure data collected during the testing stage. An important class of SRGMs that has been widely studied and used by researches is NHPP SRGM that incorporates the testing-effort functions (TEF). Quite a lot of SRGMs based on NHPP that incorporates the TEF as mentioned in section 3.1 of chapter 3.

This chapter illustrates how to integrate a Log-Logistic testing-effort function into inflection S-shaped NHPP growth models (Ohba, 1984; 1984a) to get a better description of the software fault detection phenomenon. The parameters of the model are estimated by Least Square Estimation (LSE) and Maximum Likelihood

Estimation (MLE) methods. The experiments are performed based on real data sets and the results are compared with other existing models. The experimental results show that the proposed SRGMs with Log-Logistic testing-effort function can estimate the number of initial faults better than that of other models and that the Log-Logistic testing-effort functions is suitable for incorporating into inflection S-shaped NHPP growth models. Further, the analyses of the proposed models under imperfect debugging environment are also discussed.

4.2 Software Reliability Growth Model with TEF

A software reliability growth model (SRGM) explains the time dependent behavior of fault removal. The objective of software reliability testing is to determine probable problems with the software design and implementation as early as possible to assure that the system meets its reliability requirements. Several software reliability growth models have been developed over the years. Exponential growth model and inflection S-shaped growth model have been shown to be very useful in fitting software failure data.

4.3 Log-Logistic Testing-Effort Function

Testing effort is the key element in the framework of software testing. Substantial testing-effort (number of test cases, human power and CPU time) is consumed during the software debugging phase. The testing-effort indicates how the faults are detected effectively in the software and can be modeled by different distributions (Musa *et al*. 1987; Yamada *et al*. 1986; 1993; Kapur *et al*. 1999). Gokhale and Trivedi (1998) proposed the log-logistic SRGM that can capture the increasing/decreasing nature of the failure occurrence rate per fault. Recently, Bokhari and Ahmad (2006) also presented how to use the log-logistic curve to describe the time-dependent behavior of testing effort consumptions during testing.

The Cumulative testing-effort expenditure consumed in (0, *t*] is depicted in the following:

$$W(t) = \alpha[1 - \{1 + (\beta t)^{\delta}\}^{-1}]$$

$$= a[(\beta t)^{\delta} / (1 + (\beta t)^{\delta})],\ t > 0\ . \tag{32}$$

Therefore, the current testing-effort expenditure at testing t is given by:

$$w(t) = \frac{dW(t)}{dt} = \alpha\left[0 + \ 1 + (\beta t)^{\delta} \ \ ^{-2}\right] \times \delta(\beta t)^{\delta - 1} \times \beta$$

$$= \alpha \ 1 + (\beta t)^{\delta} \ \ ^{-2} \times \delta\beta(\beta t)^{\delta - 1},$$

$$w(t) = [\alpha\beta\delta(\beta t)^{\delta - 1}] / [1 + (\beta t)^{\delta}]^{2},\ t > 0, \alpha > 0, \beta > 0, \delta > 0, \tag{33}$$

where α is the total amount of testing-effort consumption required by software testing, β is the scale parameter, and δ is the shape parameter.

4.4 Inflection S-Shaped SRGM with Log-Logistic TEF

The inflection S-shaped NHPP software reliability growth model is known as one of the flexible SRGMs that can depict both exponential and S-shaped growth curves depending upon the parameter values (Ohba, 1984 & Kapur *et al.* 2004) .The model has been shown to be useful in fitting software failure data. Kapur *et al.* (2004) modified the inflection S-shaped model and incorporated the testing-effort in an NHPP model. Recently, Bokhari and Ahmad (2006) also proposed a new SRGM with the Log-Logistic testing-effort function to predict the behavior of failure and fault of a software product. In this section it is shown that how to incorporate Log-Logistic testing-effort function into inflection S-shaped NHPP model.

The extended inflection S-shaped SRGM with Log-Logistic testing-effort function is formulated on the following assumptions (Ohba, 1984; 1984a; Yamada and Osaki, 1985; Yamada *et al.* 1986; 1993; Kapur *et al.* 1999; Kuo *et al.* 2001;

Huang, 2005; Huang *et al*. 2007; Kapur *et al*. 2004; Bokhari and Ahmad, 2007; Ahmad *et al*. 2008; 2008a):

- The fault removal process follows the NHPP.
- The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
- The mean number of faults detected in the time interval $(t,\ t+\Delta t]$, $dm(t)/dt$ by the current testing-effort expenditure, $w(t)$, is proportional to the mean number of remaining faults in the software.
- The proportionality increases linearly with each additional fault removal.
- The time-dependent behavior of TEF can be modeled by a Log-Logistic distribution.
- Each time a failure occurs, the fault caused the failure is immediately removed and no new faults are introduced.
- Faults present in the software are of two types: mutually independent and mutually dependent.

There are two types of fault:

◊ The mutually independent faults which lie on different execution paths.
◊ The mutually dependent faults that lie on the same execution path.

Thus, the second type of faults is detectable if and only if faults of the first type have been removed. According to these assumptions, if the fault detection rate with respect to current testing-effort expenditures is proportional to the number of detectable faults in the software and the proportionality increases linearly with each additional fault removal, the following differential equation is obtained:

$$\frac{dm(t)}{dt}\times\frac{1}{w(t)}=\phi(t)\ \ a-m(t) \qquad (34)$$

where

$$\phi(t)=b\left[r+(1-r)\frac{m(t)}{a}\right],$$

$r\,(>0)$ is the inflection rate and represents the proportion of independent faults present in the software, $m(t)$ be the mean value function of the expected number of faults detected in time $(0,t]$, $w(t)$ is the current testing-effort expenditure at time t, a is the expected number of faults in the system, and b is the fault detection rate per unit testing-effort at time t.

Solving (34) with the initial condition that, at $t=0, W(t)=0, m(t)=0$, the following mean value function is obtained:

$$m(t)=\frac{a\left[1-e^{-bW(t)}\right]}{1+((1-r)/r)e^{-bW(t)}}\,. \tag{35}$$

The derivation of equation (35) from equation (34) is same as the derivation of equation (4) from equation (3) shown in section 3.4 in chapter 3.

If the inflection rate r = 1, the above NHPP model becomes equivalent to the exponential growth model and is given as

$$m(t)=a\left[1-e^{-bW(t)}\right].$$

The failure intensity at testing time t of the inflection S-shaped NHPP model with testing-effort is given by

$$\lambda(t)=\frac{dm(t)}{dt}=\frac{a\cdot b\cdot w(t)\cdot e^{-bW(t)}}{r\left[1+((1-r)/r)e^{-bW(t)}\right]^{2}}\,.$$

Furthermore, a flexible SRGM with mean value function is described considering the Log-Logistic testing-effort expenditure as

$$m(t)=\frac{a\left[1-e^{-b\alpha[(\beta t)^{\delta}/(1+(\beta t)^{\delta})]}\right]}{1+((1-r)/r)e^{-b\alpha[(\beta t)^{\delta}/(1+(\beta t)^{\delta})]}}\,. \tag{36}$$

On the other hand, if let the inflection rate $r = 1$ in (36), then the mean value function of SRGM is

$$m(t) = a(1 - e^{-b\alpha[(\beta t)^{\delta} / (1+(\beta t)^{\delta})]}) .$$

In addition, the expected number of faults to be detected eventually is

$$m(\infty) = \frac{a\left[1 - e^{-b\alpha}\right]}{1 + ((1-r)/r)e^{-b\alpha}} .$$

4.5 Imperfect- software-debugging models

An NHPP model is said to have perfect debugging when $a(t)$ is constant, i.e., no new faults are introduced during the debugging process. An NHPP SRGM subject to imperfect debugging was introduced by Pham (1993) with the assumption that if detected faults are removed, then there is a possibility that new faults with a constant rate γ are introduced. In this section the imperfect debugging of proposed SRGM is discussed.

Let $n(t)$ be the number of faults to be eventually detected plus the number of new faults introduced to the system by time t, the following system of differential equations is obtained.

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = \phi(t) \;\; n(t) - m(t) \quad , \tag{37}$$

$$\frac{dn(t)}{dt} = \gamma \frac{dm(t)}{dt} , \tag{38}$$

where $\phi(t) = b\left[r + (1 - r)\frac{m(t)}{n(t)} \right]$ and $n(0) = a$.

Solving the above differential equations under the boundary conditions $m(0)=0$ and $W(0)=0$, the following mean value function of inflection S-shaped model with Log-Logistic testing-effort under imperfect debugging can be obtained.

$$m_{debug}(t)=\frac{a\left[1-e^{-b(1-r\gamma)W(t)}\right]}{1-\gamma\ +\ (1-r)/r\ e^{-b(1-r\gamma)W(t)}}\,. \tag{39}$$

The derivation of equation (39) from equation (37) is same as the derivation of equation (27) from equation (25) as shown in section 3.7 of chapter 3.

Also have

$$n(t)=\frac{a\left[1+\ \ (1-r)/r\ -\gamma\ e^{-b(1-r\gamma)W(t)}\right]}{1-\gamma\ +\ (1-r)/r\ e^{-b(1-r\gamma)W(t)}}\,. \tag{40}$$

Thus, the failure intensity function $\lambda(t)$ is given by

$$\lambda(t)=\frac{ab\ 1-r\gamma\ ^{2}\ w(t)e^{-b(1-r\gamma)W(t)}}{r\left[(1-\gamma)+\ (1-r)/r\ e^{-b(1-r\gamma)W(t)}\right]^{2}}\,. \tag{41}$$

The expected number of remaining faults after testing time t is

$$m_{remaining}(t)=n(t)-m(t)=\frac{a(1-r\gamma)e^{-b(1-r\gamma)W(t)}}{r(1-\gamma)+(1-r)e^{-b(1-r\gamma)W(t)}}\,. \tag{42}$$

4.6 Estimation of Model Parameters

The parameters of the SRGM are estimated based upon the failure data using Maximum Likelihood estimation (MLE) and Least Square estimation (LSE) techniques (Musa *et al*. 1987; Musa, 1999; Lyu, 1996; Ahmad *et al*. 2008). The performance of the proposed model is then compared with other existing models. Experiments on three real software failure data are performed.

4.6.1 Least Square Estimation

The parameters α, β, and δ in the Log-Logistic testing-effort function is estimated using the method of Least Square estimation. These parameters are determined for n observed data pairs in the form (t_k, W_k) $(k=1,2,....,n;$ $0 < t_1 < t_2 < .... < t_n)$, where W_k is the cumulative testing-effort consumed in time $(0, t_k]$. The estimators $\hat{\alpha}, \hat{\beta}$, and $\hat{\delta}$, which contribute the model with a greater fitting, are obtained by minimizing (Bokhari and Ahmad, 2006; Ahmad *et al*. 2008; 2009) the following equations:

$$\ln W_k = \ln \alpha = \delta \ln(\beta t_k) - \ln[1 + (\beta t_k)^{\delta} ,$$

$$S(\alpha, \beta, \delta) = \sum_{k=1}^{n} [\ln W_k - \ln \alpha - \delta \ln(\beta t_k) + \ln[1 + (\beta t_k)^{\delta}]^2 . \tag{43}$$

Differentiating S with respect to α , β , and δ , setting the partial derivatives to zero, the following set of nonlinear equations are obtained.

$$\frac{\partial S}{\partial \alpha} = 2\sum_{k=1}^{n} \Big[\ln W_k - \ln \alpha - \delta \ln(\beta t_k) + \ln \ \ 1 + (\beta t_k)^{\delta} \ \ \Big] . \frac{-1}{\alpha} = 0 ,$$

$$\sum_{k=1}^{n} \ln W_k - n \ln \alpha - \delta \sum_{k=1}^{n} \mathrm{n}(\beta t_k) + \sum_{k=1}^{n} \ln[1 + (\beta t_k)^{\delta} = 0 ,$$

$$\sum_{k=1}^{n} \ln W_k - \delta \sum_{k=1}^{n} \mathrm{n}(\beta t_k) + \sum_{k=1}^{n} \ln[1 + (\beta t_k)^{\delta} = n \ln \alpha ,$$

$$\frac{\sum_{k=1}^{n} \ln W_k - \delta \sum_{k=1}^{n} \mathrm{n}(\beta t_k) + \sum_{k=1}^{n} \ln[1 + (\beta t_k)^{\delta}}{n} = \ln \alpha .$$

Thus, the LSE of α is given by

$$\hat{\alpha} = e^{\sum_{k=1}^{n} \ln W_k - \delta \sum_{k=1}^{n} \ln(\beta t_k) + \sum_{k=1}^{n} \ln(1 + (\beta t_k)^{\delta}) \Big/ n} . \tag{44}$$

Similarly the LSE of β and δ can be obtained by the following equations:

$$\frac{\partial S}{\partial \beta} = 2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\delta t_k}{\beta t_k} + \frac{\delta t_k(\beta t_k)^{\delta-1}}{1+(\beta t_k)^{\delta}}\right] = 0,$$

$$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\delta}{\beta} + \frac{\beta\delta t_k{}^{\delta}\beta^{\delta-1}}{\beta\ 1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\delta}{\beta} + \frac{\delta t_k{}^{\delta}\beta^{\delta}}{\beta\ 1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\delta\ 1+(\beta t_k)^{\delta}\ +\delta\ \ \beta t_k\ {}^{\delta}}{\beta\ 1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\delta}{\beta\ 1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $$2\frac{-\delta}{\beta}\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{1}{1+(\beta t_k)^{\delta}}\right] = 0,$$

$$\frac{\partial S}{\partial \beta} = \sum_{k=1}^{n}[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln[1+(\beta t_k)^{\delta}].\left[\frac{1}{(1+(\beta t_k)^{\delta})}\right] = 0\ . \tag{45}$$

$$\frac{\partial S}{\partial \delta} = 2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[-\ln(\beta t_k) + \frac{(\beta t_k)^{\delta}\ln(\beta t_k)}{1+(\beta t_k)^{\delta}}\right] = 0$$

,

or, $$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big]$$

$$\times\ \left[\frac{-\ln(\beta t_k)(1+(\beta t_k)^{\delta}) + (\beta t_k)^{\delta}\ln(\beta t_k)}{1+(\beta t_k)^{\delta}}\right] = 0\ ,$$

or, $$2\sum_{k=1}^{n}\Big[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln\ 1+(\beta t_k)^{\delta}\ \Big].\left[\frac{-\ln(\beta t_k)}{1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $$-2\sum_{k=1}^{n}[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln[1+(\beta t_k)^{\delta}].\left[\frac{\ln(\beta t_k)}{1+(\beta t_k)^{\delta}}\right] = 0,$$

or, $\sum_{k=1}^{n}[\ln W_k - \ln\alpha - \delta\ln(\beta t_k) + \ln[1+(\beta t_k)^{\delta}].\left[\dfrac{\ln(\beta t_k)}{1+(\beta t_k)^{\delta}}\right] = 0$. (46)

4.6.2 Maximum Likelihood Estimation

The estimates of the parameters of the SRGM are estimated through Maximum Likelihood Estimation method. The estimators of $a, b,$ and r are determined for the n observed data pairs in the form (t_k, y_k) $(k = 1, 2, \ldots, n;\ 0 < t_1 < t_2 < \ldots\ldots < t_n)$ where y_k is the cumulative number of software faults detected up to time t_k or $(0, t_k]$. Then the likelihood function for the unknown parameters $a, b,$ and r in the NHPP model (36) is given by (Musa *et al.* 1987).

$$L'(a,b,r) \equiv P\ \ N(t_i) = y_i,\ \ i = 1, 2, \ldots, n$$

$$= \prod_{k=1}^{n} \frac{[m(t_k) - m(t_{k-1})]^{(y_k - y_{k-1})}}{(y_k - y_{k-1})!} . e^{-[m(t_k) - m(t_{k-1})]}, \quad (47)$$

where $t_0 \equiv 0$ and $y_0 \equiv 0$.

Taking logarithm on both sides in (47), the following is obtained:

$$L = \ln L' = \sum_{k=1}^{n} (y_k - y_{k-1}) \ln [m(t_k) - m(t_{k-1})]$$

$$- \sum_{k=1}^{n} [m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n} \ln [(y_k - y_{k-1})!].$$

Now

$$m\ \ t_k\ \ - m\ \ t_{k-1}\ \ = \frac{a\ \ 1+\lambda\ \ \ e^{-bW\ \ t_{k-1}}\ \ - e^{-bW\ \ t_k}}{1+\lambda e^{-bW\ \ t_k}\ \ \ 1+\lambda e^{-bW\ \ t_{k-1}}},$$

and have

$$\sum_{k=1}^{n}\left[m\ \ t_k\ \ - m\ \ t_{k-1}\ \ \right] = m\ \ t_n\ \ = \frac{a\left[1 - e^{-bW\ \ t_n}\ \right]}{1+\lambda e^{-bW\ \ t_n}},$$

Thus,

$$L = \sum_{k=1}^{n}\ y_k - y_{k-1}\ \ \ln a + \sum_{k=1}^{n}\ y_k - y_{k-1}\ \ \ln\ \ 1+\lambda\ \ + \sum_{k=1}^{n}\ y_k - y_{k-1}\ \ \ln\ \ e^{-bW\ \ t_{k-1}}\ \ - e^{-bW\ \ t_k}$$

$$-\sum_{k=1}^{n} y_k - y_{k-1} \;\ln\; 1+\left[\; 1-r \;\;/r\right]e^{-bW\;t_k} \quad -\sum_{k=1}^{n} y_k - y_{k-1} \;\ln\; 1+\left[\; 1-r \;\;/r\right]e^{-bW\;t_{k-1}}$$

$$-\frac{a\left[1-e^{-bW\;t_n}\;\right]}{1+\left[\;1-r\;\;/r\right]e^{-bW\;t_n}}-\sum_{k=1}^{n}\ln\left[\;y_k-y_{k-1}\;\;!\right]. \tag{48}$$

The maximum likelihood estimates of SRGM parameters a, r and b can be obtained by solving the following equations.

$$\frac{\partial L}{\partial a}=\frac{\sum_{k=1}^{n}\;y_k-y_{k-1}}{a}-\frac{a\left[1-e^{-bW(t_n)}\right]}{1+\;(1-r)/r\;\;e^{-bW(t_n)}}=0\,,$$

$$\frac{y_n}{a}=\frac{\left[1-e^{-bW(t_n)}\right]}{1+\;(1-r)/r\;\;e^{-bW(t_n)}}\,,$$

$Let\ \phi_n = e^{-bW(t_n)}$.

$$\hat{a}=\frac{1+[(1-r)/r]\;\;\phi_n}{1-\phi_n}\;\;. \tag{49}$$

$$\frac{\partial L}{\partial r}=\frac{\sum_{k=1}^{n}\;y_k-y_{k-1}}{1+\;(1-r)/r}\times\left(-\frac{1}{r^2}\right)-\sum_{k=1}^{n}\frac{y_k-y_{k-1}}{1+\;(1-r)/r\;\;e^{-bW(t_k)}}\times\left(-\frac{1}{r^2}e^{-bW(t_k)}\right)$$

$$-\sum_{k=1}^{n}\frac{(y_k-y_{k-1})}{1+\;(1-r)/r\;\;e^{-bW\;t_{k-1}}}\times\left(-\frac{1}{r^2}e^{-bW\;t_{k-1}}\;\right)-\frac{a\left[1-e^{-bW\;t_n}\;\right]}{\left[1+\;(1-r)/r\;\;e^{-bW(t_n)}\right]^2}\times\left(-\frac{1}{r^2}e^{-bW(t_n)}\right),$$

where $\phi_k = e^{-bW(t_k)},\ k=1,2,\ldots n,$ and $\lambda=\dfrac{1-r}{r}$

$$\frac{\partial L}{\partial r}=\frac{-y_n}{r^2(1+\lambda)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_k}{r^2(1+\lambda\phi_k)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_{k-1}}{r^2(1+\lambda\phi_{k-1})}-\frac{a\phi_n\;\;1-\phi_n}{r^2(1+\lambda\phi_n)^2}=0$$

$$\frac{y_n}{r^2(1+\lambda)}+\frac{a\phi_n\ \ 1-\phi_n}{r^2(1+\lambda\phi_n)^2}=\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_k}{r^2(1+\lambda\phi_k)}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})\phi_{k-1}}{r^2(1+\lambda\phi_{k-1})}=0 \quad , \qquad (50)$$

$$\frac{\partial L}{\partial b}=\sum_{k=1}^{n}\frac{y_k-y_{k-1}}{\left[e^{-bW\ \ t_{k-1}}\ \ -e^{-bW\ \ t_k}\ \ \right]}\times\ -W\ \ t_{k-1}\ \ e^{-bW\ \ t_{k-1}}\ \ +W\ \ t_k\ \ e^{-bW\ \ t_k}$$

$$-\sum_{k=1}^{n}\frac{y_k-y_{k-1}}{1+\lambda e^{-bW\ \ t_k}}\times\ -W\ \ t_k\ \ \lambda e^{-bW\ \ t_k}$$

$$-\sum_{k=1}^{n}\frac{y_k-y_{k-1}}{1+\lambda e^{-bW\ \ t_{k-1}}}\times\ -W\ \ t_{k-1}\ \ \lambda\ \ e^{-bW\ \ t_{k-1}}$$

$$-\frac{aW\ \ t_n\ \ e^{-bW\ \ t_n}\ \ 1+\lambda}{\left[1+\lambda e^{-bW\ \ t_n}\ \ \right]^2}=0\,,$$

$$\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(W(t_k)\phi_k-W(t_{k-1})\phi_{k-1}}{[\phi_{k-1}-\phi_k]}+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(\lambda W(t_k)\phi_k)}{1+\lambda\phi_k}$$

$$+\sum_{k=1}^{n}\frac{(y_k-y_{k-1})(\lambda W(t_{k-1})\phi_{k-1})}{1+\lambda\phi_{k-1}}=\frac{aW(t_n)\phi_n(1+\lambda)}{1+\lambda\phi_n\ \ {}^2}\quad . \qquad (51)$$

The above can be solved by numerical methods to get the values of $\hat{a}$, $\hat{b}$ and $\hat{r}$.

Finally, if the sample size n of (t_k, y_k) is sufficiently large, then the maximum-likelihood estimates $\hat{a}$, $\hat{b}$ and $\hat{r}$ asymptotically follow a trivariate s-normal (TVN) distribution (Yamada *et al*. 1993; Yamada and Osaki, 1985; Kapur *et al*. 1999; 2004; Huang and Kuo, 2002). That is,

$$\begin{pmatrix}\hat{a}\\ \hat{b}\\ \hat{r}\end{pmatrix}\sim T\text{VN}\left[\begin{pmatrix}\hat{a}\\ \hat{b}\\ \hat{r}\end{pmatrix}\ ,\ V\right],\ \text{as}\ n\to\infty\,. \qquad (52)$$

The variance-covariance matrix V in the asymptotic properties of (52) is useful in qualifying the variability of the estimated parameters $\hat{a}$, $\hat{b}$ and $\hat{r}$, and is the inverse of the Fisher information matrix F, i.e., $V=F^{-1}$ (Yamada *et al*. 1986;

1993; Yamada and Osaki, 1985), given by the expectation of the negative of the second partial derivative of L as

$$F = \begin{bmatrix} E\left[-\frac{\partial^2 L}{\partial a^2}\right] & E\left[-\frac{\partial^2 L}{\partial a \partial b}\right] & E\left[-\frac{\partial^2 L}{\partial a \partial r}\right] \\ E\left[-\frac{\partial^2 L}{\partial a \partial b}\right] & E\left[-\frac{\partial^2 L}{\partial b^2}\right] & E\left[-\frac{\partial^2 L}{\partial b \partial r}\right] \\ E\left[-\frac{\partial^2 L}{\partial a \partial r}\right] & E\left[-\frac{\partial^2 L}{\partial b \partial r}\right] & E\left[-\frac{\partial^2 L}{\partial r^2}\right] \end{bmatrix}. \tag{53}$$

The asymptotic variance- covariance matrix V of the maximum-likelihood estimates for $\hat{a}$, $\hat{b}$ and $\hat{r}$ is the inverse of the Fisher information matrix.

$$V = F^{-1} = \begin{bmatrix} Var(\hat{a}) & Cov(\hat{a},\hat{b}) & Cov(\hat{a},\hat{r}) \\ Cov(\hat{a},\hat{b}) & Var(\hat{b}) & Cov(\hat{b},\hat{r}) \\ Cov(\hat{a},\hat{r}) & Cov(\hat{b},\hat{r}) & Var(\hat{r}) \end{bmatrix}. \tag{54}$$

The $(1-\alpha)100\%$ confidence limits for a, b and r is obtained as (Yamada and Osaki, 1985):

$$\hat{a} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{a})} \le a \le \hat{a} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{a})}\,, \tag{55}$$

$$\hat{b} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{b})} \le b \le \hat{b} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{b})} \tag{56}$$

and

$$\hat{r} - z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{r})} \le r \le \hat{r} + z_{\alpha/2}\sqrt{\mathrm{Var}(\hat{r})}\,, \tag{57}$$

where $z_{\alpha/2}$ is the $(1-\alpha/2)$ quartile of the standard normal distribution.

4.7 Data Analysis and Model Comparison

4.7.1 Comparison Criteria

To check the performance of SRGM with Log-Logistic testing-effort function, the following four criteria are used.

1. The Accuracy of Estimation (AE) is defined (Musa *et al*. 1987; Yamada and Osaki, 1985; Huang and Kuo, 2002; Kuo *et al*. 2001) as

$$AE = \left| \frac{M_a - a}{M_a} \right| ,$$

where M_a is the actual cumulative number of detected faults after the test, and *a* is the estimated number of initial faults. For practical purposes, M_a is obtained from software fault tracking after software testing.

2. The Mean of Squared Errors (MSE) (Long-term predictions) is defined (Lyu, 1996; Huang and Kuo, 2002: Kuo *et al*. 2001) as

$$MSE = \frac{1}{k} \sum_{i=1}^{k} \quad m(t_i) - m_i \quad ^2 ,$$

where $m(t_i)$, is the expected number of faults at time t_i estimated by a model, and m_i is the expected number of faults at time t_i. MSE gives a quantitative comparison for long-term predictions. A smaller MSE indicates a minimum fitting error and better performance (Huang *et al*. 1997; Kapur and Younes, 1996; Kapur *et al*. 1999).

3. The coefficient of multiple determination is defined (Musa *et al*. 1987; Musa, 1999) as

$$R^2 = \frac{S \;\; \hat{\hat{\alpha}}, 0, 1 \;\; - S \;\; \hat{\alpha}, \hat{\beta}, \hat{\delta}}{S \;\; \hat{\hat{\alpha}}, 0, 1} ,$$

where $\hat{\hat{\alpha}}$ is the LSE of α for the model with only a constant term, that is, $\beta = 0$, and $\delta = 1$ in (6). It is given by $\ln \hat{\hat{\alpha}} = \frac{1}{n} \sum_{k=1}^{n} \ln W_k$. Therefore, R^2

measures the percentage of total variation about the mean accounted for by the fitted model and tells us how well a curve fits the data. It is frequently employed to compare models and assess which model provides the best fit to the data. The best model is the one which provides the higher R^2, that is, closer to 1 (Kumar *et al*. 2005; Ahmad *et al*. 2008). To investigate whether a significant trend exists in the estimated testing-effort, one could test the hypotheses $H_0 : \beta = 0,$ and $\delta = 1$, against $H_1 : \beta \neq 0$ or $\delta \neq 1$ using F-test by merely forming the ratio

$$F = \frac{\left[S\ \hat{\hat{\alpha}},0,1\ - S\ \hat{\alpha},\hat{\beta},\hat{\delta}\ \right] \big/ 2}{S\ \hat{\hat{\alpha}},0,1\ \big/\ n-3}.$$

If the value of F is greater than $F_\alpha\ 2, n-3$, which is the α percentile of the F distribution with degrees of freedom 2 and $n-3$, it can be $(1-\alpha)100$ percent confident that H_0 should be rejected, that is, there is a significant trend in the testing-effort curve.

4. The Predictive Validity is defined (Musa *et al*. 1987; Musa, 1999) as the capability of the model to predict future failure behavior from present and past failure behavior. Assume that q failures by the end of test time t_q are observed. The failure data up to time $t_e\ (\leq t_q)$ is used to determine the parameters of $m(t)$. Substituting the estimates of these parameters in the mean value function yields the estimate of the number of failures $\hat{m}(t_q)$ by t_q. The estimate is compared with the actually observed number q. This procedure is repeated for several of t_e. The ratio

$$\frac{\hat{m}(t_q) - q}{q}$$

is called the relative error. Values close to zero for relative error indicate more accurate prediction and hence a better model. The predictive validity can be visually checked by plotting the relative error for normalized test time t_e / t_q.

4.7.2 Model Comparison with actual Data Set

Data Set 1

Test Time (weeks)	**Cumulative execution time (CPU hours)**	**Cumulative faults**	**Test Time (weeks)**	**Cumulative execution time (CPU hours)**	**Cumulative faults**
1	2.45	15	11	26.23	233
2	4.90	44	12	27.76	255
3	6.86	66	13	30.93	276
4	7.84	103	14	34.77	298
5	9.52	105	15	38.61	304
6	12.89	110	16	40.91	311
7	17.10	146	17	42.67	320
8	20.47	175	18	44.66	325
9	21.43	179	19	47.65	328
10	23.35	206			

The first set of actual data is from the study by Ohba (1984). The system is PL/1 data base application software, consisting of approximately 1,317,000 lines of code. During nineteen weeks of testing, 47.65 CPU hours were consumed and about 328 software faults were removed. Moreover, the total cumulative number of detected faults after a long time of testing was 358. The estimated parameters $\alpha, \beta, \text{and}\, \delta$ of the Log-Logistic testing-effort function are:

$$\hat{\alpha} = 1451.227, \qquad \hat{\beta} = 0.00256, \qquad \hat{\delta} = 1.116 \ .$$

Figure 12, shows the fitting of the estimated testing-effort by using above estimates. The fitted curves and the actual software data are shown by dotted and solid lines, respectively.

The estimated values of the parameters $a, b, \text{and}\, r$ in (35) are:

$$\hat{a} = 385.625, \qquad \hat{b} = 0.0622, \qquad \hat{r} = 0.3689 \ .$$

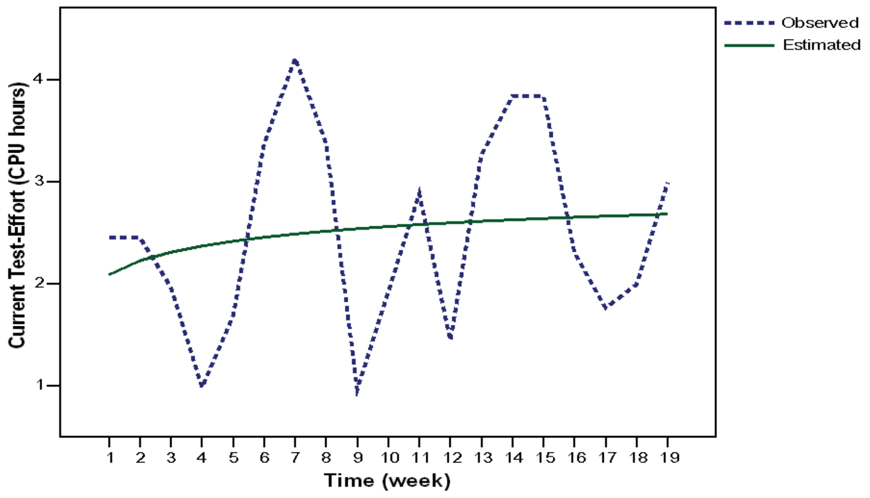

Figure 12: Observed/estimated current testing-effort function vs. time

Figure 13, illustrates a fitted curve of the estimated cumulative failure curve with the actual software data. The R^2 value for proposed Log-Logistic testing-effort is 0.99574. Therefore, it can be said that the proposed curve is suitable for modeling the software reliability. Also, the calculated value $F(=4.9787)$ is greater than $F_{0.05}(2,16)$. therefore, it can be concluded that the proposed model is suitable for modeling the software reliability and the fitted testing-effort curve is highly significant for this data set. Table V lists the comparisons of proposed model with different SRGMs which reveal that the proposed model has better performance. Kolmogorov Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

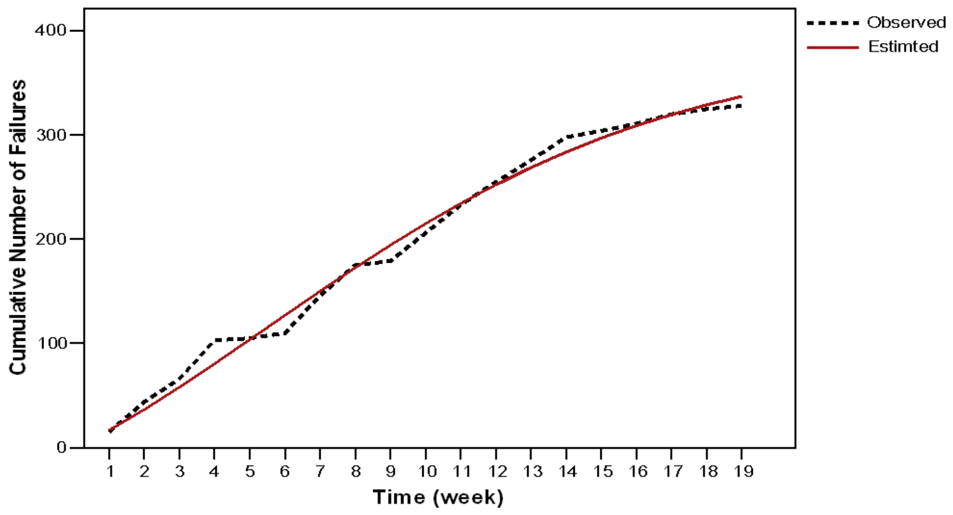

Figure 13: Observed/estimated cumulative number of failures vs. time

Table V: Comparison results of different SRGMs for Data Set 1

Model	*a*	*r*	*b*	AE %	MSE
Proposed model	385.63	0.37	0.06223	7.54	87.69
Bokhari Log-Logistic model [6]	565.73		0.01964	58.02	116.74
Yamada Rayleigh model	459.08		0.0273367	28.23	268.42
Yamada Weibull model	565.35		0.01965	57.91	122.09
Yamada delayed S-shaped model	374.05		0.197651	4.48	168.67
Delayed S-Shaped with Logistic TEF	346.55		0.0936	3.20	147.61
Inflection S-shaped model	389.1	0.2	0.0935493	8.69	133.53
Huang Logistic model	394.08		0.04272	10.06	118.59
G-O model	760.0		0.0322688	112.29	139.82

Finally, the relative error in prediction of proposed model for this data set is calculated and illustrated by Figure 14. It is observed that relative error approaches zero as t_e approaches t_q and the error curve is usually within ± 5 percent. Therefore, Figures 12 to 14 and Table V reveal that the proposed model has better performance than the other models.

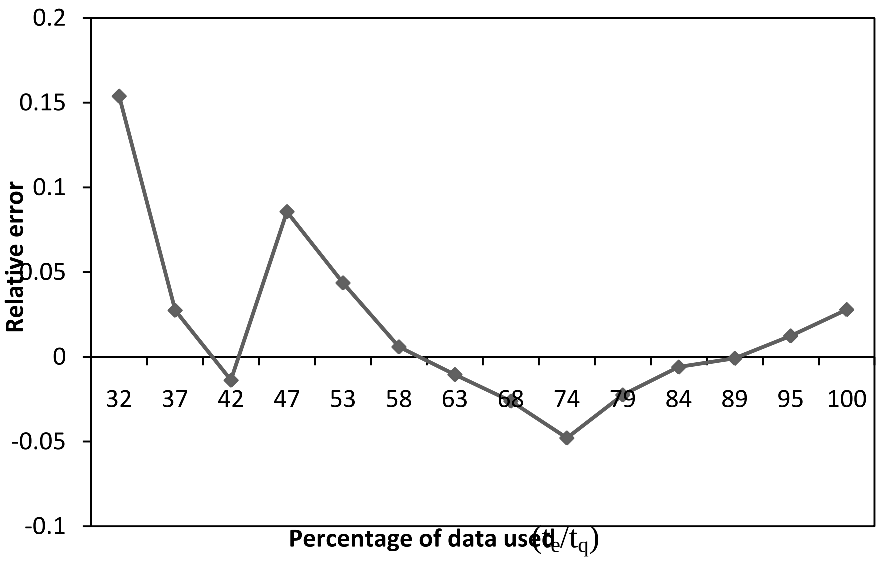

Figure 14: Predictive Relative Error Curve

Data Set 2:

Test Time (weeks)	**Cumulative execution time (CPU hours)**	**Cumulative faults**	**Test Time (weeks)**	**Cumulative execution time (CPU hours)**	**Cumulative faults**
1	519	16.00	11	6539	81.00
2	968	24.00	12	7083	86.00
3	1430	27.00	13	7487	90.00
4	1893	33.00	14	7846	93.00
5	2490	41.00	15	8205	96.00
6	3058	49.00	16	8564	98.00
7	3625	54.00	17	8923	99.00
8	4422	58.00	18	9282	100.00
9	5218	69.00	19	9641	100.00
10	5823	75.00	20	10000	100.00

The second set of actual data relates to the Release of Tandem Computer Project cited in Wood (1996) from a subset of products for four separate software releases. In this research only Release 1 is used for illustrations. There were 10000 CPU hours consumed, over the 20 week of testing and 100 software faults were removed.

The estimated parameters of the testing-effort function using LSE method are obtained as:

$$\hat{\alpha} = 15808.074, \qquad \hat{\beta} = .0704, \qquad \hat{\delta} = 1.569.$$

Figure 15 illustrates the comparisons between the observed failure data and the estimated Log-Logistic testing-effort data. Here, the fitted curves are shown as a dotted line and the solid line represents actual software data. The estimated values of the parameters of the SRGM (35) using MLE are:

$$\hat{a} = 197.4934, \qquad \hat{b} = 1.05 \times 10^{-5}, \qquad \hat{r} = 9.934 .$$

Figure 16 illustrates a fitted curve of the estimated cumulative failure curve with the actual software data. The R^2 value for proposed Log-Logistic testing-effort is 0.99757, which is very close to one. Moreover, the value of MSE is 18.04, which is very small compared to other SRGM as listed in the Table VI.

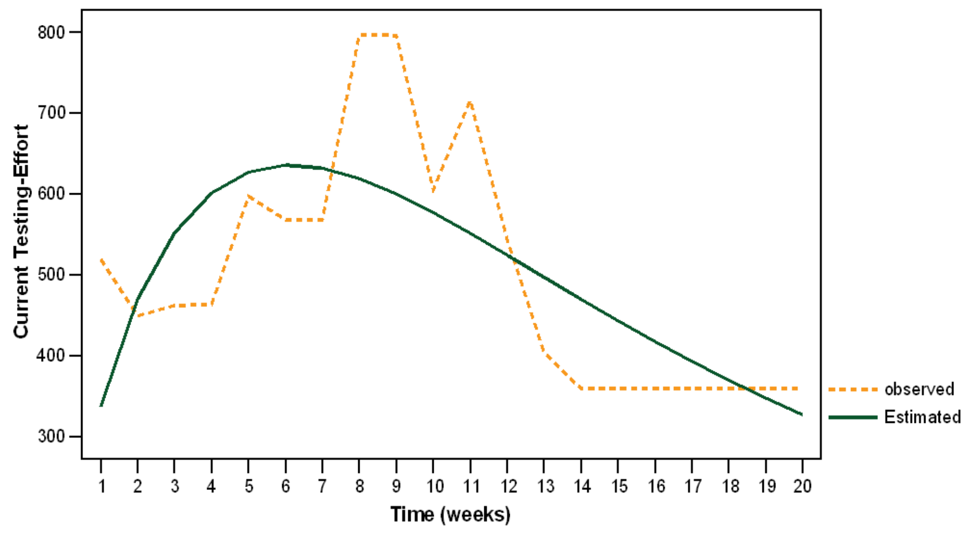

Figure 15: Observed/estimated current testing effort vs. time

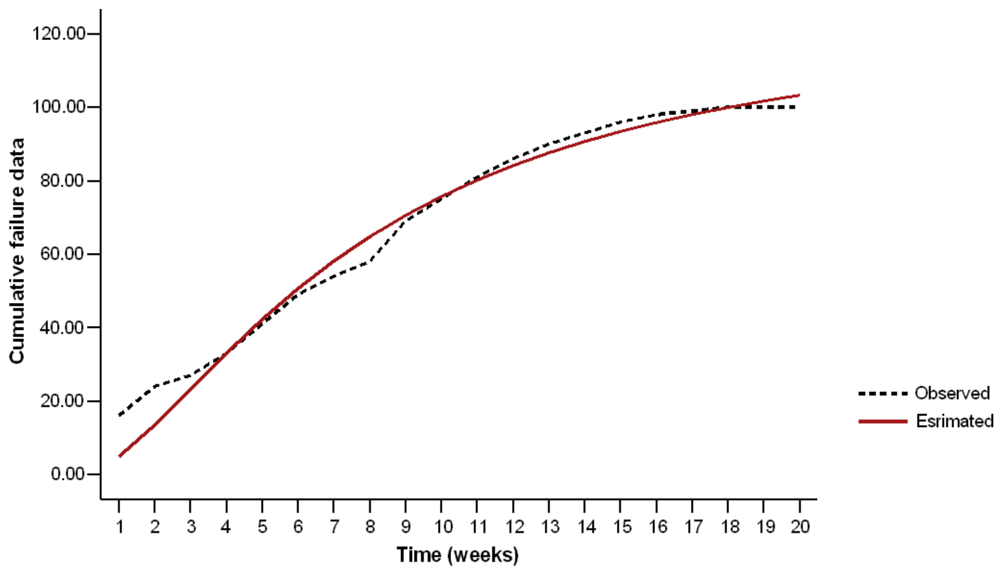

Figure 16: Observed/estimated cumulative number of failures vs. time

The comparison of proposed model with different SRGMs is listed in Table VI. It is revealed that the proposed model has better performance and therefore the Log-Logistic testing-effort function is suitable for modeling the proposed SRGM of this data set. Also the fitted testing-effort curve is significant since the calculated value $F(=5.3204)$ is greater than $F_{0.05}(2,17)$. Kolmogorov Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

Table VI. Comparison results of different SRGMs for Data Set 2

Model	***a***	***r***	***b***	**MSE**
Proposed model	197.49	9.934	0.0000105	18.04
Bokhari Log-Logistic [6]	135.91		0.000142	19.80
Huang Logistic model	107.66		0.000266	22.76
Yamada Rayleigh model	110.60		0.000226	39.69
Yamada Weibull model	135.74		1.42E-04	18.43
Yamada delayed S-shaped model	102.26		0.345	94.99
Delayed S-shaped model with Logistic TEF	101.86		0.000635	92.66
Inflection S-Shaped model	507.63	1.695	0.00814	87.45
G-O model	137.072		0.0515445	25.33

Following the work of Musa *et al.* (1987), the relative error in prediction for this data set is computed and results are plotted in Figure 17. Figures 15 to 17 and Table VI, illustrate that the proposed model has better performance and predicts the future behavior well.

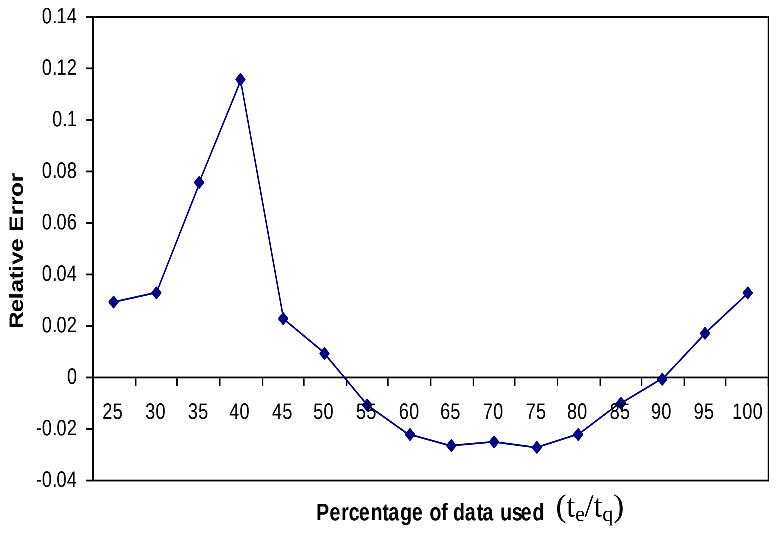

Figure 17: Predictive Relative Error Curve

Data Set 3:

Test Time (weeks)	**Cumulative execution time (CPU hours)**	**Cumulative faults**	**Test Time (weeks)**	**Cumulative execution time (CPU hours)**	**Cumulative faults**
1	.00917	2	12	2.11017	44
2	.01917	2	13	3.05017	55
3	.02217	2	14	4.39017	69
4	.04517	3	15	7.71017	87
5	.08617	4	16	11.27017	99
6	.09017	6	17	13.93017	111
7	.11517	7	18	17.70017	126
8	.41717	16	19	21.10017	132
9	1.39017	29	20	23.50017	135
10	1.41017	31	21	25.30017	136
11	1.86017	42			

The third set of actual data in this research is the System T1 data of the Rome Air Development Center (RADC) projects and cited from Musa *et al.* (1987), Musa (1999). The number of object instructions in System T1 which is a real-time command and control application is 21,700. The software was tested by nine testers over the period of 21 weeks. Through the testing phase, about 25.3 CPU hours were consumed and 136 software faults ere removed. The number of faults removed after 3.5 years of test was reported to be 188 (Huang, 2005).

Using LSE method the estimated parameters of the Log-Logistic testing-effort function are:

$$\hat{\alpha} = 33.1105, \qquad \hat{\beta} = 0.0565, \qquad \hat{\delta} = 7.151 \ .$$

The estimated values of the SRGM (35) parameters by MLE are:

$$\hat{a} = 161.024, \qquad \hat{b} = 0.00106, \qquad \hat{r} = 168.369 \ .$$

Figure 18 shows the fitting of the Log-Logistic testing-effort whereas Figure 19 illustrates a fitted curve of the estimated cumulative number of failures.
The fitted curves and the actual software data are shown by dotted and solid lines, respectively.

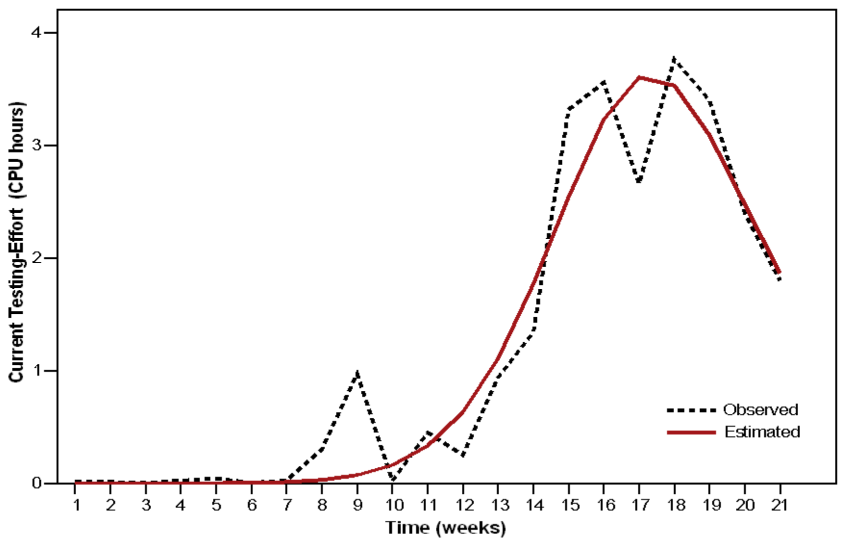

Figure 18: Observed/estimated current testing-effort function vs. time

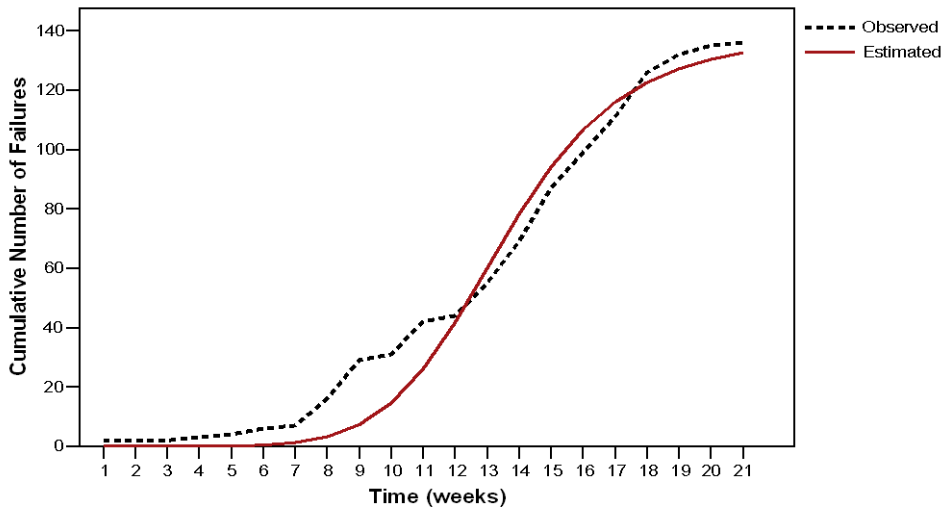

Figure 19: Observed/estimated cumulative number of failures vs. time

The R^2 value for proposed Log-Logistic testing-effort is 0.9973. Therefore, it can be said that the proposed curve is suitable for modeling the software reliability. Also, the calculated value $F(=5.6512)$ is greater than $F_{0.05}(2,18)$ and $F_{0.01}(2,18)$, which concludes that the fitted testing-effort curve is highly suitable for this data set. Table VII lists the comparisons of proposed model with different SRGMs which reveal that the proposed model has fairly better performance. Kolmogorov

Smirnov goodness-of-fit test shows that the proposed SRGM fits pretty well at the 5 percent level of significance.

Table VII. Comparison results of different SRGMs for Data Set 3

Model	***a***	***r***	***b***	**AE**(%)	**MSE**
Proposed model	160.26	21.85	0.00107	14.75	75.22
Bokhari Log-Logistic model [6]	133.28		0.1571	29.11	100.18
Yamada Rayleigh model	866.94		0.00962	25.11	89.241
Yamada Weibull model	133.71		0.155	28.88	81.51
Yamada delayed S-shaped model	237.196		0.09635	26.16	245.25
Delayed S-Shaped model with Logistic TEF (Huang)	124.11		0.411	33.98	180.02
Inflection S-shaped model	159.11		0.0765	15.36	118.3
Huang Logistic model	138.03		0.14509	26.58	64.41
G-O model	142.32		0.1246	24.29	2438.3

Lastly, the relative error in prediction of proposed model for this data set is calculated and plotted against the percentage of data used (that is, t_e / t_q) in Figure 20. Consequently, from the Figures 18 to 20 and Tables VII, it can be concluded that the proposed model gets reasonable prediction in estimating the number of software faults and fits the observed data moderately better than the others.

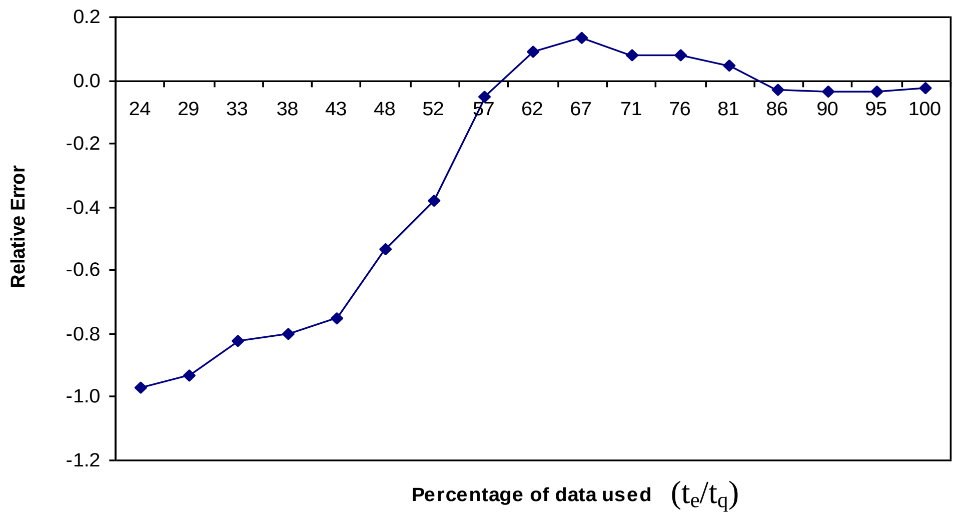

Figure 23: Predictive Relative Error Curve

4.7.3 Numerical Example in an Imperfect Debugging Environment.

In order to validate the proposed SRGM under imperfect debugging, MSE is selected as the evaluation criterion. To discuss the issue of imperfect debugging for proposed SRGM, Data Set 1 is taken into account. The parameters $a, b, r,$ and γ in (39) can be solved by the method of MLE. Table VIII shows the estimated parameters of the proposed SRGM and some selected models for comparison under imperfect debugging. It also gives the results of MSE. It is observed that the value of MSE of the proposed SRGM with Log-logistic testing-effort function is the lowest among all the models considered. Moreover, the estimated values of γ of all models is close to but not equal to zero, thus it can be concluded that the fault removal phenomenon in the software development may not be pure perfect debugging process (Huang *et al*. 2000). A fitted curve of the estimated cumulative number of failures with the actual software data and the RE curve for the proposed SRGM with Log-Logistic testing-effort function under imperfect debugging is illustrated by Figures 24 and 25.

Table VIII. Comparison results of different SRGMs for Data Set 1 under Imperfect Debugging

Models	***a***	***r***	***b***	γ	MSE
Proposed model	372.67	0.377	0.0630	0.0336	87.69
Bokhari Log-Logistic model [6]	565.73		0.01964	58.02	116.74
Huang Logistic model	391.62		0.0420	0.0116	114.09
Yamada Rayleigh model	399.02		0.0316	0.123	268.55
Yamada Weibull model	565.35		0.01965	57.91	122.09
Yamada delayed S-shaped	19.87		1.68	0.962	114.71
Delayed S-Shaped model with Logistic TEF (Huang)	335.39		0.124	0.0115	634.60
Inflection S-Shaped model	256.37	0.34	0.201	0.330	98.21
G-O model	530.61		0.0463	0.287	222.09

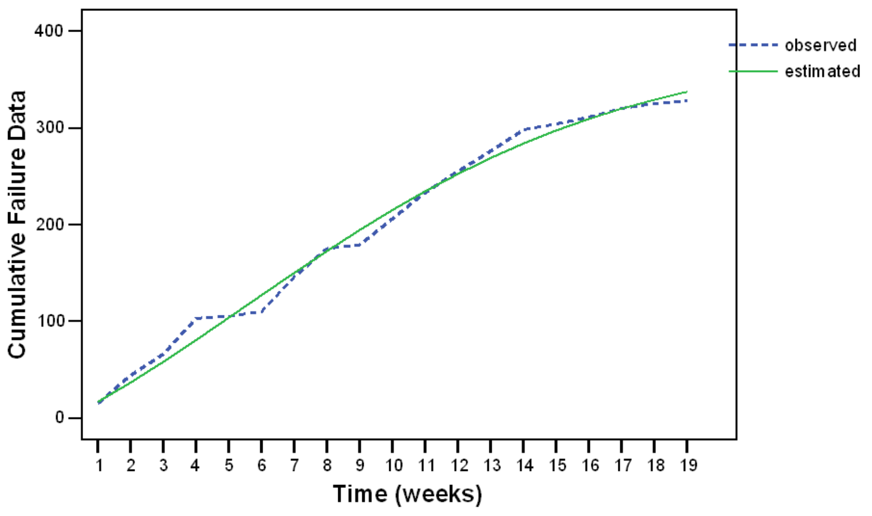

Figure 24: Observed/estimated cumulative number of failures vs. time

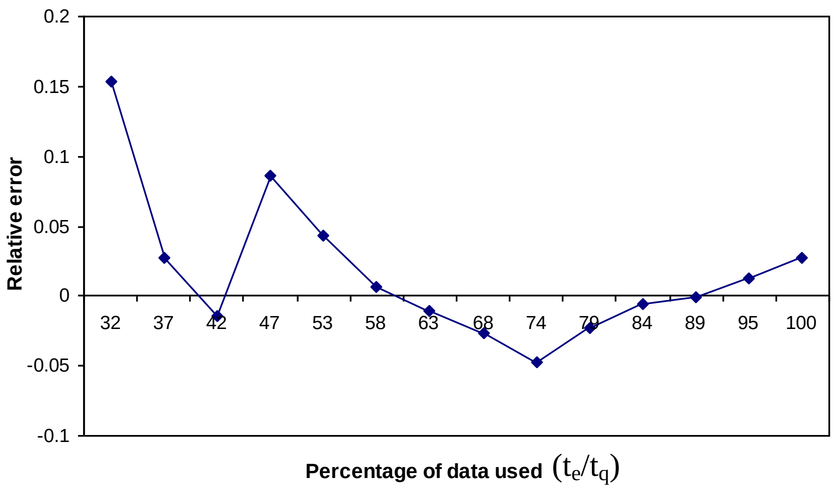

Figure 25: Predictive Relative Error Curve

CHAPTER 5: Determining the Optimal Allocation Testing Resource for Modular Software System using Dynamic Programming

5.1 Introduction

Software that plays a vital role in the modern life is broadly classified as operating system and applications software. Software is created by human and, therefore, a high degree of reliability cannot be guaranteed to perform without fault. Thus, software reliability becomes a crucial feature of the computer systems and the breakdown in the system could result in the fiscal, possessions, and human loss. Although, measuring or predicting software reliability is a difficult task, it is important for the assessment of the performance of the underlying software systems. In addition to software reliability measurement, SRGMs can also help to predict the fault detection coverage in the testing phase. In the field of software reliability, many SRGMs have been proposed over the decades as mentioned in section 1.3.3 of chapter 1.

A software development process consists of four phases: specification, design, coding and testing (Myers, 1976). Testing is the final phase where the software is tested to detect and correct software faults in the following consecutive stages: module testing, integration testing, system testing and installation testing. Software consists of several modules that are tested independently during the module testing, and the interfaces among these modules are tested in the integration testing. Lastly, the complete software system of which modules are interconnected is tested under the stimulated user environment in the system testing.

Moreover, all the testing activities of different modules should be completed with a limited resource, approximately 40% - 50% of the total amount of software development resources (Yamada *et al*. 1995). When resources available are limited in the testing of a software, it is important to allocate the testing resources efficiently among all the modules so that the maximum reliability of the complete system is achieved. As a fact, the reliability of the software is indirectly proportional to the number of faults removed; the problem of maximization of the software reliability may be treated as an equivalent problem of the maximization of number of faults to be removed. Many authors have addressed this problem to determine the optimal resource allocation over the years, including Ohtera and Yamada (1990); Yamada *et al*. (1995); Leung (1997); Xie and Yang (2001); Lo *et al*. (2002); Lyu *et al*. (2002); Huang *et al*. (2002 & 2004); Kapur *et al*. (2004); Huang and Lyu (2005); and Huang and Lo (2006).

In this chapter, four strategies of optimal resource allocation problems for module testing are discussed and formulated as nonlinear programming problems (NLPP). The first NLPP maximizes the total number of faults expected to be removed during module testing satisfying the available testing resources. Sometimes management may want a desire level of reliability to be achieved or/and a certain percentage of number of faults to be removed. Adding these requirements to the first NLPP as constraints the other NLPPs are formulated. These NLPPs are modeled by the inflection S-shaped SRGM based on a Poisson process (NHPP) with exponentiated Weibull (EW) testing-effort function. Treating a testing module as a stage, the NLPPs are considered as multi-stage decision problems and solved by developing a solution procedure using dynamic programming technique. It is shown that the optimal allocation of testing-resources among software modules can improve software reliability. Finally, numerical examples are given to illustrate the procedure developed in this chapter and the results are compared with that of Kapur *et al*. (2004). It reveals that the proposed dynamic programming method within our frame work yields a gain in efficiency over Kapur *et al*. (2004).

5.2 Problem of Testing Resource Allocation

In modular software system the testing-resource has to be allocated appropriately to each software module, which is tested independently and simultaneously, so that the maximum reliability of the complete system is achieved. In this section, four kinds of testing-resource allocation problems are considered to make the best use of a specified total testing-resource for flexible SRGM with EW testing-effort functions as discussed in sections 3.4 and 3.5 of chapter 3.

Assumptions (Yamada *et al*. 1995 ; Xie and Yang 2001; Huang and Lyu, 2005; Huang and Lo, 2006; Khan *et al*. 2008):

1) The software system is composed of N independent modules, and the software modules are tested independently. The number of software faults remaining in each module can be estimated by the SRGM with EW testing effort function.
2) The total amount of testing-resource expenditures for the module testing is specified.
3) The manager has to allocate the specified total testing-resource expenditures to each software module wisely so that the software faults to be removed in the system may be maximized and the desired software reliability level is achieved.

5.2.1 Formulation of Resource Allocation Problems as NLPPs

Suppose that a software has N modules to be tested independently for removing faults in the whole testing phase. Let a_i and b_i be the expected number of initial faults content and the fault detection rate, respectively in ith module $i = 1, 2, ..., N$, which are already estimated by the SRGM discussed in Section 3.4. Also, let W_i be the amount of resources to be used in ith module during a fixed testing time T, where W_i is given as:

$$W_i = \alpha_i \; 1 - e^{-\beta_i t^{\delta_i}} \;^{\theta_i}, \; \alpha_i > 0, \; \beta_i > 0, \; \delta_i > 0, \; \theta_i > 0 .$$

Then, from the Section 3.4, the SRGM with EW testing resource for ith module can be written as:

$$m_i = \frac{a_i \; 1 - e^{-b_i W_i}}{1 + \left[\; 1 - r_i \; / r_i \right] e^{-b_i W_i}}, \tag{58}$$

where m_i be the number of faults expected to be removed from the ith module during a fixed testing time T and r_i be the proportion of independent faults.

It can be noticed that (58) is a nonlinear function of W_i and may be expressed as:

$$m_i = \frac{a_i \;\; 1-e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}}\,, \tag{59}$$

where

$$\lambda_i = \frac{1-r_i}{r_i}\,. \tag{60}$$

If C be the total testing resources available for the whole testing phase, then the problem of optimum resource allocation may formulated as the following NLPPs.

NLPP-I:

If the software project manager's aim is only to determine the optimum allocation of the testing resources to each of module that maximizes the number of faults removed in the system, then a reasonable criterion is to maximize the sum of the weighted total number of faults removed, that is,

$$\text{Maximize } Z = \sum_{i=1}^{N} v_i m_i = \sum_{i=1}^{N} \frac{v_i a_i \;\; 1-e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}}\,,$$

where v_i are positive weight assigned to ith module according to its relative importance of fault removal. If $v_i = 1$ for all $i = 1, 2, ..., N$, the objective is to maximize the total number of faults removed.

One may choose these weights such that v_i is proportional to the number of initial faults a_i. The basis of this choice is the fact that the given system may be heterogeneous with respect to a module's initial faults a_i. If a module has large a_i, then m_i of the module should also expected to be large. Therefore, a way to maximize Z is to assign more weight to m_i which could be done by using the given conjecture.

With $\sum_{i=1}^{N} v_i = 1$, this choice gives

$$v_i = \frac{a_i}{\sum_{i=1}^{N} a_i} \; ; \; i = 1, 2, ..., N . \tag{61}$$

With a linear resource function "$\sum_{i=1}^{N} W_i$" the problem of finding the optimum allocation of the testing resources to each of module for a fixed resource C may be given as the following NLPP:

$$\text{Maximize } Z = \sum_{i=1}^{N} v_i m_i = \sum_{i=1}^{N} \frac{v_i a_i \; 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}} ,$$

$$\text{Subject to } \sum_{i=1}^{N} W_i \le C ,$$

$$W_i \ge 0 \; ; \; i = 1, 2, ..., N . \tag{62}$$

NLPP-II:

Sometimes in the above allocation procedure one or more modules may not get any resources, to which the management may not agree where the modules are not tested any further. Moreover, during module testing it is also expected that each module is tested satisfactorily so that a certain percentage of the fault content is desired to be removed from each module of the software. Consequently, the NLPP in (62) needs to be modified to maximize the removal of the faults in the software under resource constraint subject to the desired level of faults to be removed from each of the modules in the software.

Let p_i be the minimum proportion of initial faults to be removed in ith module, that is

$$m_i \ge p_i a_i .$$

This, along with (59), gives

$$W_i \geq A_i \,, \tag{63}$$

where A_i is the minimum amount of resources required to be allocated to ith module to remove p_i proportion of initial faults and is given by

$$A_i = -\frac{1}{b_i} \ln\left(\frac{1-p_i}{1+p_i\lambda_i} \right). \tag{64}$$

Adding (63) as a constraint, NLPP in (62) reduces to

$$\begin{aligned} \text{Maximize } \; & Z = \sum_{i=1}^{N} \frac{v_i a_i \left(1-e^{-b_i W_i}\right)}{1+\lambda_i e^{-b_i W_i}}, \\ \text{Subject to } \; & \sum_{i=1}^{N} W_i \leq C \,, \\ & W_i \geq A_i \,; \\ & A_i \geq 0 \,;\; i = 1, 2, \ldots, N \,. \end{aligned} \tag{65}$$

Note that the NLPP governed by (65) has a feasible optimum solution if and only if $\sum_{i=1}^{N} A_i \leq C$. Otherwise, the management requires additional resources of amount $\sum_{i=1}^{N} A_i - C$ to remove the desired p_i proportion of initial faults. Further, from (64) it may be noted that $A_i = 0$, if $p_i = 0$. This implies that if there is no restriction on the proportion of the faults to be removed, NLPP-II in (65) reduces to NLPP-I in (62).

NLPP-III:

The management may also aspire that the allocation of testing resource should also achieve at a certain level of reliability for each of the modules.

Let R_0 be the minimum level of reliability to be achieved for the modules, that is

$$R_i \geq R_0$$

where

$$R_i(t) = \frac{a_i \;\; 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}}\,.$$

This gives

$$W_i \geq B_i\,, \tag{66}$$

where B_i is the minimum amount of resources required to be allocated to ith module to achieve R_0 level of reliability of removing faults from the modules and is given by

$$B_i = -\frac{1}{b_i}\ln\left(\frac{1 - R_0}{1 + R_0\lambda_i}\right). \tag{67}$$

Therefore, adding (66) as a constraint, the NLPP-I is modified to maximize the removal of the faults in the software subject to the resource constraint and the desired level of reliability to be achieved for the modules, and may be expressed as

$$\begin{aligned} \text{Maximize } \; Z &= \sum_{i=1}^{N} \frac{v_i a_i \;\; 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}}\,, \\ \text{Subject to } \; &\sum_{i=1}^{N} W_i \leq C\,, \\ &W_i \geq B_i\,; \\ &B_i \geq 0\,;\; i = 1, 2, ..., N\,. \end{aligned} \tag{68}$$

As discussed earlier this NLPP too has a feasible optimum solution if and only if $\sum_{i=1}^{N} B_i \leq C$. Otherwise, an additional resource of amount $\sum_{i=1}^{N} B_i - C$ is required to achieve R_0 level of reliability. Also note that $B_i = 0$ when $R_0 = 0$. This implies that if there is no restriction on the level of reliability to be achieved, NLPP-III in (68) reduces to NLPP-I.

NLPP-IV:

When both restrictions (63) and (66) are to be attended simultaneously, the problem of optimum resource allocation to maximize the removal of the faults under resource constraint subject to the desired level of faults to be removed as well as the desired level of reliability to be achieved from each of the modules may be written as

$$\text{Maximize } \; Z = \sum_{i=1}^{N} \frac{v_i a_i \;\; 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}} \, ,$$

$$\text{Subject to } \; \sum_{i=1}^{N} W_i \le C \, ,$$

$$W_i \ge A_i \, ;$$

$$W_i \ge B_i \, ;$$

$$A_i \ge 0 \text{ and } B_i \ge 0 \, ; \; i = 1, 2, ..., N \qquad (69)$$

This NLPP has a feasible optimum solution if and only if $\max \;\; \sum_{i=1}^{N} A_i ,\; \sum_{i=1}^{N} B_i \;\; \le C$. Otherwise, an additional resource of amount $\max \;\; \sum_{i=1}^{N} A_i , \sum_{i=1}^{N} B_i \;\; - C$ is required to meet the restrictions. Also note that $p_i = 0$ and $R_0 = 0$ give $A_i = 0$ and $B_i = 0$ respectively. Thus, NLPP-IV in (69) generalizes all the NLPPs in (62), (65) and (68). In the following section the solution procedures for the NLPP in (69) and then subsequently for NLPPs in (62), (65) and (68) are discussed.

5.3 Solution Procedure

The objective function and the constraints of the NLPPs (I-IV) are sums of separable functions of W_i; $i = 1, 2, ..., N$. A function $f(x_1, x_2, ..., x_N)$ is said to be separable if it can be expressed as $f(x_1, x_2, ..., x_N) = \sum_{j=1}^{N} f_j(x_j)$. Due to this separability and the nature of the problem, the dynamic programming technique may be used to solve the NLPPs (see Hadley (1970), Khan *et al.* (1997, 2003, 2008, 2008a)).

Consider the kth subproblem of NLPP in (69), involving the first k molules:

$$\text{Maximize } Z_k = \sum_{i=1}^{k} \frac{v_i a_i \;\; 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}},$$

$$\text{Subject to } \sum_{i=1}^{k} W_i \le C_k,$$

$$W_i \ge A_i;$$

$$W_i \ge B_i; \; i = 1, 2, ..., k.$$

$$A_i \ge 0 \text{ and } B_i \ge 0; \; i = 1, ..., k \qquad (70)$$

where C_k denotes the amount of resources available for the first k modules. Note that $C_k < C$ if $k < N$ and $C_N = C$.

Also $\quad C_k = W_1 + W_2 + ... + W_k$

and $\quad C_{k-1} = W_1 + W_2 + ... + W_{k-1} = C_k - W_k$; $i = 1, 2, ..., k$. $\qquad$ (71)

Let $z_k \;\; C_k$ denote the maximum value of the objective function of the problem (70), then

$$z_k \;\; C_k = \max\left\{ Z_k \;\middle|\; \sum_{i=1}^{k} W_i \le C_k;\; W_i \ge C_i;\; W_i \ge D_i;\; i = 1, 2, ..., k \right\}. \qquad (72)$$

With this definition of $z_k\ \ C_k$ the NLPP (70) is equivalent to $z_N(C)$, which can be obtained by finding $z_k\ \ C_k$ recursively for $k = 1, 2, ..., N$ and for all feasible C_k , $0 \le C_k \le C$.

Further, (72) can be written as

$$z_k\ \ C_k\ \ = \max\left\{ \frac{v_k a_k\ \ 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + \sum_{i=1}^{k-1} \frac{v_i a_i\ \ 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}} \middle| \sum_{i=1}^{k-1} W_i \le C_k - W_k ; W_i \ge A_i ; W_i \ge B_i ; i = 1, ..., k \right\}.$$

For a fixed value of W_k, $\max\ \ A_k, B_k\ \ \le W_k \le C_k$, $z_k\ \ C_k$ is given by

$$z_k\ \ C_k\ \ = \frac{v_k a_k\ \ 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + \max\left\{ \sum_{i=1}^{k-1} \frac{v_i a_i\ \ 1 - e^{-b_i W_i}}{1 + \lambda_i e^{-b_i W_i}} \middle| \sum_{i=1}^{k-1} W_i \le C_k - W_k ; W_i \ge A_i ; W_i \ge B_i ; i = 1, ..., k-1 \right\}. \qquad (73)$$

By the definition (72) the quantity inside in (73) is $z_{k-1}\ \ C_{k-1}$, where $C_{k-1} = C_k - W_k$.

Thus, the required recurrence relation for solving the NLPP (69) is

$$z_k\ \ C_k\ \ = \max_{\max\ A_k, B_k\ \le W_k \le C_k} \left(\frac{v_k a_k\ \ 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1}\ \ C_{k-1} \right), \qquad (74)$$

Also $z_k\ \ C_k\ \ = 0$ for $k = 0$ and $z_k\ \ C_k\ \ = 0$ are defined if

$$W_k < \max\ \ A_k, B_k\ \ \text{or } W_k > C_k ;\ k = 1, 2, ..., N . \qquad (75)$$

This takes care of the restrictions $W_i \ge A_i$ and $W_i \ge B_i$; $i = 1, 2, ..., N$ of the NLPP (69).

The algorithm of the above solution procedure is summarized in Appendix A. At the final stage of the solution, i.e. at $k = N$, $z_N(C)$ is obtained by solving (74) recursively for all C_k. From $z_N(C)$ the optimum value W_N^* of W_N is obtained, from $z_{N-1}(C_{N-1})$ the optimum value W_{N-1}^* of W_{N-1} is obtained and so on until finally the optimum value W_1^* of W_1 is obtained.

The recurrence relation for the NLPP-I in (62) is obtained in a similar way as

$$z_k \;\; C_k \; = \max_{0 \le W_k \le C_k} \left(\frac{v_k a_k \;\; 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1} \;\; C_{k-1} \right). \tag{76}$$

Also $z_k \;\; C_k \; = 0$ for $k = 0$ and $z_k \;\; C_k \; = 0$ if $W_k > C_k$; $k = 1, 2, ..., N$. (77)

The recurrence relation for the NLPP-II in (65) is

$$z_k \;\; C_k \; = \max_{A_k \le W_k \le C_k} \left(\frac{v_k a_k \;\; 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1} \;\; C_{k-1} \right). \tag{78}$$

Define $z_k \;\; C_k \; = 0$ for $k = 0$ and $z_k \;\; C_k \; = 0$ if

$$W_k < A_k \text{ or } W_k > C_k \text{ ; } k = 1, 2, ..., N . \tag{79}$$

This takes care of the restrictions $W_i \ge A_i$; $i = 1, 2, ..., N$ of the NLPP-II.

Finally, the recurrence relation for the NLPP-III in (68) is

$$z_k \;\; C_k \; = \max_{B_k \le W_k \le C_k} \left(\frac{v_k a_k \;\; 1 - e^{-b_k W_k}}{1 + \lambda_k e^{-b_k W_k}} + z_{k-1} \;\; C_{k-1} \right). \tag{80}$$

Define $z_k \;\; C_k \; = 0$ for $k = 0$ and $z_k \;\; C_k \; = 0$ if

$$W_k < B_k \text{ or } W_k > C_k \text{ ; } k = 1, 2, ..., N , \tag{81}$$

which takes care of the restrictions $W_i \ge B_i$; $i = 1, 2, ..., N$ of the NLPP-III.

5.4 Numerical Examples and Experimental Results

In this section, several examples are discussed to demonstrate the use of proposed method and to show how the optimal allocation of testing resource to each module is determined. All the model parameters a_i, b_i and r_i for each module where estimated by using the maximum likelihood estimation (MLE) and weighted least square estimation (WLSE) methods discussed in section 3.5 of Chapter 3 and the software failure data are summarized in Table IX (Kapur *et al.* 2004; Khan *et al.* 2008). The total resource taken for the testing is 110,000 units.

Table IX: Estimated value of SRGM parameters

Module	a_i	b_i	r_i	$\lambda_i = \frac{1-r_i}{r_i}$
1	45	0.000412932	0.85412	0.1707957
2	13	0.000319987	0.885236	0.1296423
3	16	0.000264216	0.889588	0.1241159
4	35	0.000150122	0.789985	0.2658468
5	14	8.95707E-05	0.795781	0.2566271
6	21	5.87323E-05	0.754922	0.3246402
7	20	3.20165E-05	0.58921	0.6971878
8	11	3.15115E-05	0.578634	0.7282082
Total	175			

It is assumed that there is no restriction on the number of faults to be removed and the reliability to be achieved, that is $A_i = 0$ and $B_i = 0$. With this assumptions the required problem of finding optimum resource allocation is represented by the NLPP-I. Three kinds of weighting vectors v_i, are considered. They are equal weights, the weights as proposed in (61) and the arbitrary weights as suggested by Huang and Lyu (2005) and Haung and Lo (2006). These weighting vectors are listed in Table X.

Table X: Weights corresponding to each module

Module	Arbitrary v_i	Equal v_i	Proposed v_i
1	1.0	1	0.3
2	0.5	1	0.1
3	0.6	1	0.1
4	0.4	1	0.2
5	0.9	1	0.1
6	0.2	1	0.1
7	0.4	1	0.1
8	0.6	1	0.1

With the different weighting vectors the recurrence relation (76) of NLPP-I in (62) are solved separately using a computer programme developed for the solution procedure discussed in Section 5.3. The optimum allocation of testing resource W_i^* to the modules is shown in XI. Using (58), the optimum number of faults m_i^* removed along with the percentages of faults removed for each module is also presented in the table for the different weighting vectors.

Table XI: Optimal allocation of testing resource with different weights

	Arbitrary v_i			Equal v_i			Proposed v_i		
Module	W_i*	m_i^*	% removed	W_i*	m_i^*	% removed	W_i*	m_i^*	% removed
1	12495.04	45	100	10255.11	44	97.8	12777.62	45	100.0
2	9130.09	12	92.3	8405.59	12	92.3	8216.75	12	92.3
3	11800.58	15	93.8	10225.36	15	93.8	9996.50	15	93.8
4	20208.77	33	94.3	20178.14	33	94.3	24482.71	34	97.1
5	26603.27	12	85.7	16787.11	10	71.4	16041.29	10	71.4
6	11047.07	9	42.9	25753.15	15	71.4	24576.66	15	71.4
7	18715.18	7	35	18395.53	6	30	13908.46	5	25
8	0	0	0	0	0	0	0	0	0
Total	110000	133	76	109999.99	135	77.1	109999.99	136	77.7

The Table XI reveals that the maximum number of total faults is removed by the use of proposed weighting vector and the total number of faults removed from the software through this allocation is 136. That is, 77.7% of the fault content being removed. Whereas, the arbitrary and equal weighting procedures remove 133 and 135 faults with 76% and 77.1% respectively.

In all three resource allocations, it is noted that the percentage of remaining fault after allocation is higher than the fault removed in module 7 and none of the faults are removed in module 8 as no testing resource is allocated. Hence, the result may not satisfy the management.

Suppose the management desires to set the restriction that at least some percentage of initial faults p_i $(i = 1, 2, ..., 8)$ as given in Table XII should be removed. With this additional restriction, the required problem of determining the optimum resource allocation is NLPP-II as given in (65).

To solve the problem A_i, minimum amount of resources required for the ith module to remove p_i proportion of initial faults, are calculated using (64) and are presented in Table XII. Since $\sum_{i=1}^{8} A_i = 99659.48$ is less than $C = 110000$, the NLPP does not have any infeasible solution.

Therefore, solving the NLPP-II with the proposed weighting vector through the recurrence relation (78), the optimum testing resource W_i^* to the modules are obtained. This result along with the expected number of faults m_i^* removed, percentages of faults removed, and faults remaining for each module are given in the Table XII. The total number of faults that can be removed through this allocation is 124 (i.e. 70.9% of the total fault content is removed).

Table XII: Optimal allocation of testing resource with different proportion different weights

Module	a_i	p_i	A_i	v_i	W_i^*	m_i^*	% removed	% remained
1	45	0.8	4207.74	0.3	8794.65	44	97.8	2.2
2	13	0.4	1754.39	0.1	2837.32	7	53.8	46.2
3	16	0.5	2851.29	0.1	3503.39	9	56.3	43.7
4	35	0.7	9156.80	0.2	13175.38	29	82.9	17.1
5	14	0.5	9086.36	0.1	9086.36	7	50.0	50
6	21	0.6	18631.25	0.1	18631.25	13	61.9	38.1
7	20	0.6	39534.57	0.1	39534.57	12	60.0	40
8	11	0.25	14437.08	0.1	14437.08	3	27.3	72.7
Total	175		99659.48		110000	124	70.9	29.1

Further, if the management wants to allocate the testing resource that can remove faults from each module with a certain level of reliability R_0, then the optimum W_i^* is determined using NLPP-III. To achieve R_0 level of reliability of removing faults, the minimum amount of resources required at ith module, B_i, is calculated using (67).

However, with the given resource $C = 110000$, it has been seen that the maximum level of reliability that can be achieved is $R_0 = 0.54 = 54\%$ as the minimum total resource required is $\sum_{i=1}^{8} B_i = 109474.13$, which is still less than C (see Table XIII). This implies that NLPP-III results in infeasible solution, if R_0 is to be achieved more than 54%. Solving the recurrence relation in (80) the optimum allocation of testing resource W_i^* to achieve 54% level of reliability is obtained, which is shown in Table XIII.

Table XIII: Optimal allocation of testing resource with reliability $R_0 = 54\%$

Module	a_i	R_0	B_i	v_i	W_i^*	m_i^*	% removed	% remained
1	45	0.54	2094.17	0.3	2620.05	28	62.2	37.8
2	13	0.54	2638.21	0.1	2638.21	7	53.8	46.2
3	16	0.54	3184.52	0.1	3184.52	9	56.3	43.7
4	35	0.54	6066.22	0.2	6066.22	19	54.3	45.7
5	14	0.54	10118.37	0.1	10118.37	8	57.1	42.9
6	21	0.54	15971.74	0.1	15971.74	11	52.4	47.6
7	20	0.54	34234.20	0.1	34234.20	11	55.0	45.0
8	11	0.54	35166.69	0.1	35166.69	6	54.5	45.5
Total	175		109474.13		110000	99	56.6	43.4

If the software manager desires to achieve higher reliability, say $R_0 = 0.9 = 90\%$, the minimum total resource required is $\sum_{i=1}^{8} B_i = 287138.70$. That is, the extra amount of testing resource is anticipated to be 287138.7 – 110000 = 177138.7. With the new total testing resource $C = 110000 + 177138.7 = 287138.7$ the optimum allocation of testing resource W_i^* to modules through NLPP-III is obtained as shown in Table XIV.

Further allocation of testing resource can also be done whereby the developer desires simultaneously to remove a certain proportion of faults and to achieve a level of reliability as discussed in NLPP-IV. If the total amount of testing resource is $C \geq \max \sum_{i=1}^{8} A_i, \sum_{i=1}^{8} B_i$, a feasible optimum allocations W_i^* to modules can be obtained.

Table XIV: Optimal allocation of testing resource with reliability $R_0 = 90\%$

Module	a_i	R_0	B_i	v_i	W_i^*	m_i^*	% removed	% remained
1	45	0.9	5922.46	0.3	5922.46	41	91.1	8.9
2	13	0.9	7540.75	0.1	7540.75	12	92.3	7.7
3	16	0.9	9115.57	0.1	9115.57	14	87.5	12.5
4	35	0.9	16767.04	0.2	16767.04	32	91.4	8.6
5	14	0.9	28026.83	0.1	28026.83	13	92.9	7.1
6	21	0.9	43569.09	0.1	43569.09	19	90.5	9.5
7	20	0.9	87130.42	0.1	87130.42	18	90.0	10.0
8	11	0.9	89066.54	0.1	89066.54	10	90.9	9.1
Total	175		287138.70		287138.70	159	90.9	9.1

5.5 Discussion

In this section, a study of comparing the optimum resource allocation proposed in this Chapter to Kapur *et al*. (2004) allocations is made. In particular, if $v_i = 1$ and $p_i = p_0 = 0.5$ for $i = 1, 2, ..., 8$, the problems discussed in NLPP-I & II are equivalent to the formulation of Kapur *et al*. A brief review of this formulation is given below.

Using Dur *et al*. (2001) method, Kapur *et al*. treated the NLPP-I as the following multiobjective fractional programming problem:

$$\text{Maximize } Z\ W\ = \left[l_1\ W_1\ / n_1\ W_1\ , l_2\ W_2\ / n_2\ W_2\ , ..., l_N\ W_N\ / n_N\ W_N\ \right]^T ,$$

$$\text{Subject to } W \in S = \left\{ W \in R^N \Big/ \sum_{i=1}^{N} W_i \le C, W_i \ge 0, i = 1, ...N \right\} \tag{82}$$

where $l_i = a_i\ 1 - e^{-b_i W_i}$ and $n_i = 1 + \lambda_i e^{-b_i W_i}$.

For fixed weight of objective function the problem (82) is further approximated and expressed as Geoffrion (1968)'s equivalent scalarization formulation as

$$\text{Maximize } Z = \sum_{i=1}^{N} \phi_i \; l_i \; W_i \; - n_i \; W_i \quad ,$$

$$\text{Subject to } \sum_{i=1}^{N} W_i \le C \,,$$

$$W_i \ge 0 \,; \; i = 1, 2, ..., N$$

$$\phi \in \Omega = \; \phi \in R^N \big/ \sum \phi_i = 1, \phi_i \ge 0, i = 1, ..., N \; . \qquad (83)$$

The problem (83) is then solved by using dynamic programming approach for which the recurrence relations are obtained as

$$f_k \; C \; = \max_{0 \le W_k \le C} \; a_k - 1 \; - \; a_k + \lambda_k \; e^{-b_k W_k} + f_{k-1} \; C - W_k \quad ; \; i = 1, 2, ..., N \, . \; (84)$$

Differentiating (84) for each stage and proceeding by induction the closed form solutions are obtained by

$$W_i^* = \frac{1}{b_i + \mu_{i-1}} \left[\mu_{i-1} C - \ln\left(\frac{\mu_{i-1} V_{i-1}}{a_i + \lambda_i \; b_i} \right) \right], \; i = 1, 2, ..., N \,, \qquad (85)$$

where $\mu_i = \dfrac{1}{\sum_{j=1}^{i} 1/b_j}$ and $\mu_i V_i = \prod_{j=1}^{i} \Big[\; a_j + \lambda_j \; b_j \Big]^{a_j/b_j} \;, \; i = 1, 2, ..., N$.

It can be observed that the solution obtained by (84) is obviously an approximate one as the problem (68) is approximated to (83). It has also been seen that Kapur *et al*. (2004) used *C*, the whole amount of resources, as the state variable at k th stage to obtain the recurrence relation (84), which is constant for all stages. In practice, a state variable for k th is defined by $C_k < C$, which is the total amount of resources available at k th stage. The stat transformation function from $(k-1)$ th stage to k th stage is given by (71). Note $C_k = C$ for $k = N$.

In a similar way Kapur *et al*. also solved the NLPP-II. They obtained the results on optimum resource allocation W_i^* and the optimum number of faults removed m_i^* for the numerical example discussed in this paper.

Table XV shows the optimum number of faults m^*_{Proposed} and m^*_{Kapur} removed by the proposed and Kapur *et al*. formulations, respectively. From the Table XV it is observed that the proposed method is more efficient as it removes more faults than Kapur *et al*. in most of the modules. This table also shows the relative gain (RG) in percentage of faults removed due to the proposed method over Kapur *et al*. for all the modules. The over all relative gain (RG) in percentage of the proposed method over Kapur *et al*. is 7.1% and 11.2% respectively for NLPP-I and II.

Table XV: Comparison results

Module	NLPP-I			NLPP-II		
	m^*_{Kapur}	m^*_{Proposed}	RG in %	m^*_{Kapur}	m^*_{Proposed}	RG in %
1	40	44	10.0	33	42	27.3
2	10	12	20.0	8	8	0.0
3	13	15	15.4	9	10	11.1
4	28	33	17.9	23	25	8.7
5	9	10	11.1	7	7	0.0
6	13	15	15.4	11	11	0.0
7	9	6	-33.3	10	10	0.0
8	4	0	-100.0	6	6	0.0
Total	126	135	7.1	107	119	11.2

Where

$$\text{RG} = \frac{m^*_{\text{Proposed}} - m^*_{\text{Kapur}}}{m^*_{\text{Kapur}}}.$$

CHAPTER 6 CONCLUSION

As the cost of software application failures grows and these failures increasingly impact businesses, software reliability will become progressively more important. Employing effective software reliability engineering techniques to improve product and process reliability would be the industry's best interests as well as major challenges Many researches have been done and will continue in the field of software reliability as the demand for failure-free software is of great concern in this globalization of the world. Many industrial and commercial processes are governed by innovative software these days, and it is becoming increasingly important for software companies to develop reliable software. In this thesis, a flexible SRGM based on NHPP model is proposed, which incorporates EW testing-effort function and log-Logistic testing-effort functions into inflection S-shaped model. The SRGM model used here is Inflection S-Shaped model which was first studied by Ohba.

The performance of the proposed SRGM is compared with other traditional SRGMs using different criteria. The results obtained show better fit and wider applicability of the proposed model on different types of real data applications. It is concluded that the proposed flexible SRGM has better performance as compare to the other SRGMs and gives a reasonable predictive capability for the real failure data. Also conclude that the incorporated EW and Log-Logistic testing-effort functions into inflection S-shaped model is a flexible and can be used to describe the actual expenditure patterns more faithfully during software development. In addition, the proposed models under imperfect debugging environment are also discussed.

Moreover, four strategies on the optimal testing resource allocation problems for modular software system have been developed. The first NLPP maximizes the total number of faults expected to be removed given a fixed amount of testing resource. The second NLPP maximizes the removal of the faults given the amount of testing resource and the desired level of faults to be removed from each of the modules. The third NLPP maximizes the removal of faults given the amount of testing resource and a reliability objective for the modules. And the fourth NLPP maximizes the removal of faults given the amount of testing-resource with desired level of faults and reliability to be achieved for each of the modules.

Also, discussed are several numerical examples on resource allocation problems for module testing to show the applications and impacts of proposed method using dynamic programming technique. Based on the experimental results, it is concluded that the proposed strategies may be helpful for software project managers to make the best decisions in solving these problems. In addition, a comparison of optimum resource allocation in this research is made with that of Kapur *et al*. (2004) allocations. It is seen that using the proposed method removes more faults than Kapur *et al*. Thus, the proposed method is more competent.

REFERENCES

[1] Ahmad, N., Bokhari, M.U., Quadri, S.M.K. and Khan, M.G.M. (2008), “The Exponentiated Weibull Software Reliability Growth Model with Various Testing-Efforts and Optimal Release Policies: A performance Analysis”, *International Journal of Quality and Reliability management*, Vol.25, no. 2, pp. 211 – 235.

[2] Ahmad, N., Khan, M.G.M. and Rafi, L.S. (2008a) “Inflection S-Shaped Software Reliability Growth Models with Testing-effort Function”, *Proceeding of the VI International Symposium on Optimization and Statistics*, Dec 29-31, Aligarh, India.

[3] Ahmad, N., Khan, M.G.M., Quadri, S.M.K. and Kumar, M. (2009), “Modeling and Analysis of Software Reliability with Burr Type X Testing-Effort and Release-Time Determination”, *Journal of Modeling in Management*, Vol. 4(1), pp. 28 – 54.

[4] Ahmad, N., Khan, M. G. M and Rafi, L. S. (2010), “A Study of Testing-Effort Dependent Inflection S-Shaped Software Reliability Growth Models with Imperfect Debugging”, *International Journal of Quality and Reliability Management,* Vol. 27 (1), (At press).

[5] Berman, O. and Cutler, M. (2004), “Resource Allocation during Tests for Optimally Reliable Software”, *Computers and Operations Research*, Vol. 31, No. 11, pp. 1847-1865.

[6] Bhalla, V.K. (1992), “Optimal Release Policies for a Flexible Software Reliability Growth Model”, *Reliability Engineering & System Safety*, Vol. 35, no. 1, pp. 49-54.

[7] Bokhari, M.U. and Ahmad, N. (2006), “Analysis of a Software Reliability Growth Models: The Case of Log-Logistic Test-effort function”,

Proceedings of the 17th International Conference on Modeling and Simulation (MS'2006), Montreal, Canada, pp. 540-545.

[8] Bokhari, M.U. and Ahmad, N. (2007), "Software Reliability Growth Modeling for Exponentiated Weibull Functions with Actual Software Failures Data", *Advances in Computer Science and Engineering: Reports and Monographs,* Vol. 2, pp. 390-396, World Scientific Publishing Company, Singapore.

[9] DeMarco, T. (1982), *Controlling Software Projects*: Management, Measurement and Estimation, Pentice-Hall.

[10] Dur, M., Horst, R. and Thoai, N.V. (2001), "Solving Sum-of-ratios Fractional Programs Using Efficient Points," *Optimization*, 41, 447-466.

[11] Geoffrion, A.M. (1968), "Proper Efficient and Theory of Vector Maximization," *Journal of Mathematical Analysis and Applications*, 22, 613-630.

[12] Goel, A.L. and Okumota, K. (1979), "Time-Dependent Error-Detection rate Model for Software Reliability and Other Performance Measures", *IEEE Transactions on Reliability*, Vol.28, No.3, pp. 206-211.

[13] Gokhale, S.S. and Trivedi, K.S. (1998), "Log-Logistic Software Reliability Growth Model", In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium (HASE '98),* Washington, DC, USA, pp.34-41.

[14] Hadley, G. (1970). *Nonlinear and Dynamic Programming*. Addison Wesley Publishing Company Inc., Sydney.

[15] Hou, R.H., Kuo, S.Y. and Chang, Y.P. (1996), "Efficient Allocation of Testing Resources for Software Module testing based on the Hyper-Geometric Distribution Software Reliability Growth Model", *Proceedings of*

the 7th International Symposium on Software Reliability Engineering, pp. 289-298.

[16] Huang, C.Y. (2005), "Cost-Reliability-Optimal Release Policy for Software Reliability Models Incorporating Improvements in Testing efficiency", *Journal of Systems and Software,* Vol. 77, no. 2, pp. 139-155.

[17] Huang, C.Y. (2005a), "Performance Analysis of Software Reliability Growth Models with Testing-Effort and Change-Point", *The Journal of Systems and Software*, Vol. 76. pp. 181-194.

[18] Huang, C.Y., and Kuo, S.Y. (2002), "Analysis of Incorporating Logistic testing-Effort Function into Software Reliability Modeling", *IEEE Transaction on Reliability,* Vol.51, No. 3, pp. 261-270.

[19] Huang, C.Y., Kuo, S.Y. and Chen, I.Y. (1997), "Analysis of a Software Reliability Growth Model with Logistic Testing-Effort Function", *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pp. 378.

[20] Huang, C.Y., Kuo, Lo, J.H., S.Y. and Lyu, M.R. (1999), "Software Reliability Modeling and Cost estimation Incorporating Testing-Effort and Efficiency", *Proceedings of the Tenth International Symposium on Software Reliability Engineering,* pp. 62.

[21] Huang, C.Y., Kuo, S.Y., Lo, J.H. and Lyu, M.R. (2000), "Effort-Index based Software Reliability Growth Models and Performance Assessment", *Proceedings of the 24th IEEE Annual International Computer Software and Applications Conference* (COMPSAC'2000), pp.454-459.

[22] Huang, C.Y., Kuo, S.Y. and Lyu, M.R. (2007), "An Assessment of Testing-effort Dependent Software Reliability Growth Models", *IEEE Transactions on Reliability*, Vol. 56, no.2, pp 198-211.

[23] Huang, C.Y. and Lo, J.H. (2006), "Optimal Resource Allocation for Cost and Reliability of Modulare Software Systems in the Testing Phase", *The Journal of Systems and Software*, Vol. 79, Issue 5, pp. 653-664.

[24] Huang, C.Y., Lo, J.H., Kuo, S.Y., and Lyu, M.R. (2002), "Optimal Allocation of Testing Resources for Modular Software Systems", *Proceedings of the Thirteenth IEEE International Symposium on Software Reliability Engineering*, pp. 129 – 138.

[25] Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R. (2004), "Optimal Allocation of Testing Resource Considering Cost, Reliability, and Testing-Effort", *Tenth IEEE Pacific Rim International Symposium on Dependable Computing*, 103 – 112.

[26] Huang, C.Y., and Lyu, M.R., (2005), "Optimal Testing Resource Allocation, and Sensitivity Analysis in Software Development", *IEEE Transaction on Reliability*, Vol. 54, No. 4, pp. 592-603.

[27] Kapur P.K., Garg R.B. and Kumar S. (1999), *Contributions to Hardware and Software Reliability*, World Scientific, Singapore.

[28] Kapur, P.K., Jha, P.C., and Bardhan, A.K. (2004), "Optimal Allocation of Testing Resource for a Modular Software", *Asia-Pacific Journal of Operational Research*, Vol. 21, no. 3, pp. 333-354.

[29] Kapur, P.K., Graver, P.S., and Younes, S. (1994), "Modeling an Imperfect Debugging Phenomenon with Testing Effort", In: *Proceedings of 5th International Symposium on Software Reliability Engineering* (ISSRE'1994), pp. 178-183.

[30] Kapur, P.K., and Younes, S. (1996), "Modelling and Imperfect Debugging Phenomenon in Software Reliability", *Microelectronics and Reliability*, Vol.36, No.5, pp. 645-650.

[31] Khan, M.G.M., Ahsan, M. J. and Jahan, N. (1997) “Compromise Allocation in Multivariate Stratified Sampling: An Integer Solution,” *Naval Research Logistics*, Vol.44, 69-79.

[32] Khan, M.G.M., Ahmad, N. and Rafi, L.S. (2008), “Optimal Testing Resource Allocation for Modular Software based on a Software Reliability Growth Model: a Dynamic Programming Approach”, *IEEE Proceedings of 2008 International Conference on Computer Science and Software Engineering*, IEEE Computer Society, Vol. 2, pp. 759 – 762.

[33] Khan, M.G.M., Khan, E.A. and Ahsan, M. J. (2003) “An Optimal Multivariate Stratified Sampling Design Using Dynamic Programming”, *Australian & New Zealand Journal of Statistics*, Vol. 45(1), 107-113.

[34] Khan, M.G.M., Nand, N. and Ahmad, N. (2008a) "Determining the Optimum Strata Boundary Points using Dynamic Programming," *Survey Methodology*, Vol. 34 (2), pp. 205 – 214.

[35] Kumar, M., Ahmad, N. and Quadri, S.M.K. (2005), “Software Reliability Growth Models and Data Analysis with a Pareto Test-effort”, *RAU Journal of Research*, Vol., 15 (1-2), pp. 124-128.

[36] Kuo, S.Y., Huang, C.Y. and Lyu, M.R. (2001), “Framework for Modeling Software Reliability, using Various Testing-Efforts and Fault-Detection Rates”, *IEEE Transitions on Reliability*, Vol. 501, No. 3, pp. 310-320.

[37] Leung, Y.W. (1997), “Dynamic Resource-Allocation for Software-Module Testing”, *Journal of Systems Software*, Vol. 37, pp. 129 – 139.

[38] Lo, Jung-Hua (2005), “An Algorithm to Allocate the Testing-Effort Expenditures Based on Sensitive Analysis Method for Software Module Systems”, *Proceedings of the IEEE Region 10 Annual International Conference*, TENCON, pp. 1 – 6.

[39] Lo, J.H. and Huang, C.Y. (2004), "Incorporating Imperfect Debugging into Software Fault Correction Process", *In: Proceeding of the IEEE Regional 10 Annual International Conference (TENCON'2004)*, Chiang Mai, Thailand, pp. 326 – 329.

[40] Lo, J.H., Huang, C.Y., Kuo, S.Y., and Lyu, M.R. (2004), "Optimal Allocation of Testing Resource Considering Cost, Reliability, and Testing-Effort", *Tenth IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 103 – 112.

[41] Lo, J.H., Kuo, S.Y., Lyu, M.R., and Huang, C.Y. (2002), "Optimal Resource Allocation and Reliability analysis for Component-based Software Applications", *Proceedings of the 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC'2002)*, pp.7 – 12.

[42] Lyu, M.R. (1996*), Handbook of Software Reliability Engineering*, McGraw Hill.

[43] Lyu, M.R., Rangarajan, S., and Van Moorsel, A.P.A. (2002), "Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development", *IEEE Transactions on Reliability*, Vol. 51, No. 2, pp. 183 – 192.

[44] Musa J.D. (1999), *Software Reliability Engineering*: More Reliable Software, Faster Development and Testing, McGraw-Hill.

[45] Musa, J.D., Iannino, A. and Okumoto, K. (1987), *Software Reliability: Measurement, Prediction and Application*, McGraw-Hill.

[46] Myers, G.J. (1976), *Software Reliability: Priniples and Practices*, John Wiley & Sons.

[47] Ohba, M. (1984), Software Reliability Analysis Model”, *IBM Journal Res. Development*, Vol. 28, No. 4, pp. 428 – 443.

[48] Ohba, M. (1984a), “Inflection S-shaped Software Reliability Growth Models”, Stochastic Models in Reliability Theory (Osaki, S. and Hatoyama, Y. Editors), pp. 144 – 162, Springer-Verlag Merlin.

[49] Ohtera, and H. Yamada, S. (1990), “Optimal Allocation and Control Problems for Software-Testing Resources”, *IEEE Transactions on Reliability*, Vol. 39, no. 2, pp.171 – 176.

[50] Parr, F.N. (1980), “ An Alternative to the Rayleigh Curve for the Software Development Effort,” *IEEE Trans. Software Engineering*, Vol. SE-6, pp. 291–296.

[51] Pham, H. (2000), *Software Reliability, Springer*-Verlag, New York.

[52] Pham, H. (1993), “Software reliability assessment: Imperfect debugging and multiple failure types in software development. EG&G-RAMM-10737, Idaho National Engineering Laboratory.

[53] Pham, H. (2007), “An Imperfect-debugging Fault-detection Dependent-parameter Software”, *International Journal of Automation and Computing*, Vol. 4, no. 4, pp. 325 – 328.

[54] Pham, H., Nordmann, L. and Zhang, X. (1999), “A general Imperfect Software Debugging Model with S-Shaped Fault Detection Rate”, *IEEE Transactions on Reliability*, Vol. R-48, no. 2, pp. 169 – 175.

[55] Quadri, S. M. K., Ahmad, N., and Khan, M. G. M. (2008), “Performance Analysis of Software Reliability Model with Burr Type X Testing-Effort and Release Time”, in *Proceedings of the 2nd National Conference on Mathematical Techniques: Emerging Paradigms for Electronics and IT Industries, MATEIT-2008*, New Delhi, India, pp. 177-187.

[56] Quadri, S. M. K., Ahmad, N., and Peer, M. A. (2008a), "Software Optimal Release Policy and Reliability Growth Modeling", *Proceedings of 2nd National Conference on Computing for Nation Development*, INDIACom-2008, India, pp. 423 – 431.

[57] Quadri, S.M.K., Ahmad, N., Peer, M.A. and Kumar, M. (2006), "Nonhomogeneous Poisson Process Software Reliability Growth Model with Generalized Exponential Testing Effort Function", *RAU Journal of Research*, Vol., 16 (1-2), pp. 159 – 163.

[58] Runarsson, T.P. (2004), *Software Reliability Models*, Lecture Notes.

[59] Shyur, H.J. (2003), "A Stochastic Software Reliability Model with Imperfect and Change Point", The Journal of Systems and Software, Vol.66, pp. 135-141.

[60] Schneidewind, N.F. (1975), "Analysis of Error Processes in Computer Software", *Proceedings of International Conference on Reliable Software*, Los Angeles, pp.337 – 346.

[61] Wood, A. (1996), "Predicting Software Reliability", *IEEE Computers*, 11, pp. 69 – 77.

[62] Xie, M. (1991), *Software Reliability Modeling*, Singapore: World Scientific

[63] Xie, M. and Yang, B. (2001), "Optimal Testing-Time Allocation for Modular Systems", *International Journal of Quality and Reliability Management*, Vol. 18, No. 8, pp. 854 – 863.

[64] Xie, M. and Yang, B. (2003), "A Study of the Effect of Imperfect Debugging on Software Development Cost", *IEEE Transaction on Software Engineering*, Vol. SE-29, no. 5, pp. 471 – 473.

[65] Yamada, S., Hishitani, J. and Osaki, S. (1993), "Software Reliability Growth with a Weibull Test-Effort: A Model & Application", *IEEE Transactions on Reliability*, Vol.42, No.1, pp. 100 – 105.

[66] Yamada, S., Ichimori, T. and Nishiwaki, M. (1995), "Optimal Allocation Policies for Testing-Resource Based on a Software Reliability Growth Model", *International Journal of Mathematical and Computer Modeling*, Vol. 22, No. 10-12, pp. 295 – 301.

[67] Yamada, S. and Ohtera, H. (1990), "Software reliability growth models for testing effort control", *European Journal of Operational Research*, Vol. 46, no. 3, pp. 343 – 349.

[68] Yamada, S. and Osaki, S. (1985), "Software Reliability Growth Modeling: Models and Applications", *IEEE Transactions on Software Engineering*, Vol. Se.11, No. 12, pp. 1431 – 1437.

[69] Yamada, S., Ohtera, H. and Narihisa, H. (1986), "Software Reliability Growth Models with Testing-Effort", *IEEE Transactions on Reliability*, Vol.R.35, No.1, pp. 19 – 23.

[70] Yamada, S., Ohtera, H. and Norihisa, H. (1987), "A Testing-effort Dependent Software Reliability Model and its Application", *Microelectronics and Reliability*, Vol. 27, no. 3, pp. 507 – 522.

[71] Yamada, S., Tokuno, K. and Osaki, S. (1992), "Imperfect Debugging Models with Fault Introduction Rate for Software Reliability Assessment", *International Journal of Systems Science*, Vol. 23, no. 12, pp. 2241 – 2252.

[72] Zeephongsekul, P. (1996), "Reliability Growth of a Software Model under Imperfect Debugging and Generation of Errors", *Microelectronics and Reliability*, Vol. 36, no. 10, pp. 1475 – 1482.

[73] Zhang, X., Teng, X and Pham, H. (2003), “Considering Fault Removal Efficiency in Software Reliability Assessment”, *IEEE Transaction on System, Man, and Cybernetics* – part A, Vol. 33, no. 1, pp. 114 – 120.

APPENDIX A

The algorithm for solving NLPP (69) is summarized as:

1. Start at $k=1$. Set $z_0\ C_0\ = 0$.
2. Determine $z_1\ C_1$, the minimum value of RHS of (74) for W_1, $\max\ A_1, B_1\ \le W_1 \le C_1$ and $0 \le C_1 \le C$.
3. Record $z_1\ C_1$ and W_1.
4. For $k \ge 2,3,...,N$, express $C_{k-1} = C_k - W_k$, and set $z_k\ C_k\ = 0$ if $W_k < \max\ A_k, B_k$ and $W_k > C_k$.
5. Determine $z_k\ C_k$ for W_k, $\max\ A_k, B_k\ \le W_k \le C_k$ and $0 \le C_k \le C$.
6. Record $z_k\ C_k$ and W_k.
7. At $k = N$, $z_N(C)$ is obtained. From $z_N(C)$ the optimum value W_N^* of W_N is obtained, from $z_{N-1}(C_{N-1})$ the optimum value W_{N-1}^* of W_{N-1} is obtained and so on until the optimum value W_1^* of W_1 from $z_1(C_1)$ is finally obtained.