SOFTWARE RELIABILITY MODELING AND RELEASE TIME DETERMINATION

LI XIANG

NATIONAL UNIVERSITY OF SINGAPORE

2011

SOFTWARE RELIABILITY MODELING AND RELEASE TIME DETERMINATION

LI XIANG
(*B. Eng., UESTC*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2011

Acknowledgements

First of all, I would like to thank my supervisor, Professor Xie Min, for his pertinent supervision and insightful suggestions throughout my research life at the university. This thesis would not have been possible without Prof. Xie’s help. It is my great honor to have the chance to study under his guidance.

Secondly, I am very much grateful to Associate Professor Ng Szu Hui. As my vice supervisor, she is always available for my questions and asking for help. I have also benefited a lot from her as her teaching assistant.

Thirdly, my appreciation goes to Professor Yang Bo from University of Electronic Science and Technology of China, with whom some of my research work is carried out jointly. I have learned a lot from the cooperation with him.

Thanks also go to faculty members, staff, seniors and juniors in our ISE department. It is my great appreciation to receive the help from you. In particular, I would like to thank all the friends in ISE Computing Lab. I really enjoy the time spending with all of you!

Finally, I would like to express my unbounded gratitude towards my parents, for their unconditional love and consistent support all along the way of my study.

Table of Contents

Summary

This thesis aims to improve software reliability modeling of software failure process, and to study its corresponding release time determination problem. These objectives are achieved by extending traditional software reliability models and decision models. Research has been conducted as follows.

Software reliability models can be classified into two categories: analytical software reliability models (ASRMs) and data-driven software reliability models (DDSRMs). Both of them are studied in this thesis. In particular, an extension on ASRMs is presented in Chapter 3. In this chapter, the modeling framework for open source software reliability is introduced, and the corresponding version-updating problem is studied as well.

Besides the research on ASRMs, improvement on DDSRMs is also carried out as shown in Chapter 4. In most existing research on DDSRMs, it is generally assumed that the current failure is correlated with the most recent consecutive failures. However, this assumption restricts the failure data analysis into a special case. In order to relax this unrealistic assumption, a generic DDSRM is developed with model mining technique. The proposed model can greatly enhance the prediction accuracy.

Developing models is not the ultimate goal of software reliability modeling. It is more important to apply these models to solve corresponding decision-making problems, and software release time determination is a typical application. In Chapter 5,

sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change points is studied. Sensitivity of the software release time is investigated through various methods, including one-factor-at-a-time approach, design of experiments and global sensitivity analysis.

Although the use of sensitivity analysis can help to find out what significant parameters are and more attention can be paid for them, it is also quite possible that no more data or information is available for us to obtain more accurate estimates of parameters. Therefore, in Chapter 6, the effect of parameter uncertainty on release time determination is investigated. A risk-based approach is proposed for release time determination with delay cost considerations. It can help management have a boarder view of the release time determination problem.

Furthermore, for software release time determination problem, most existing research formulates it as single objective optimization problems. However, these formulations can hardly describe the management’s attitude accurately. Therefore, multi-objective optimization model is developed for release time determination problem in Chapter 7. In order to solve this multi-objective optimization problem, different multi-objective optimization approaches, including trade off analysis, multi-attribute utility theory, and physical programming, are used and compared in this chapter. By comparing these approaches, management can apply them more appropriately in practice considering their own unique properties.

List of Tables

List of Figures

List of Symbols

ANN	artificial neural network
ASRM	analytical software reliability model
CDF	cumulative distribution function
DDSRM	data-driven software reliability model
DOE	design of experiment
GA	genetic algorithm
LSE	least square estimation
MAUT	multi-attribute utility theory
MLE	maximum likelihood estimation
MSE	mean square error
NHPP	non-homogeneous Poisson process
OSS	open source software
SRM	software reliability model
SRGM	software reliability growth model
SVM	support vector machine
$\lambda(t)$	failure intensity function, $\lambda(t) \equiv \frac{dm(t)}{dt}$
κ	the discount rate of testing cost over time ($0 < \kappa \leq 1$)
μ_y	the expected time to remove a fault during the testing phase
μ_w	the expected time to remove a fault during the warranty phase
c_0	the expected set-up cost for software testing
c_1	the expected cost per unit testing time

c_2	the expected cost of removing a fault during the testing phase
c_3	the expected cost of removing a fault during the operation phase
c_4	the expected cost due to software failure
$E[C(t)]$	expected total cost of software development
I	Fisher information matrix
L	the likelihood function
$a(t)$	fault content function
$b(t)$	fault detection rate function
$m(t)$	mean value function of software fault content
A_i	scale parameter in logistic function for release *i*
α_i	scale parameter in logistic function for release *i*
β_i	shape parameter in Weibull function for release *i*
γ_i	shape parameter in Weibull function for release *i*
R_i	reliability estimate for release *i*
$r(t)$	risk function which measures the risk that software cannot meet its reliability requirement due to parameter uncertainty
$R(x \mid t)$	software reliability at time *x* after it has been tested for *t* unit of time
R_0	software reliability requirement from customers
t	release time of the software
$\hat{T}$	mean value of release time based on the reliability requirement R_0
T^*	optimal software release time
$Var(\hat{T})$	variance of $\hat{T}$
Z_α	the (1-α) quantile of the standard normal distribution

Chapter 1 Introduction

With the rapid increase of applications of computer systems in industries as well as in our daily life, the reliability of computer systems has become a crucial issue. Since computer systems are also widely used in safety-critical systems such as control systems in nuclear power plants or in medical instruments, the need for high reliability is even more urgent.

Computer systems are generally composed of hardware and software, and therefore ensuring high reliability of the system involves investigating reliability of both hardware and software. Unfortunately, unlike hardware reliability assurance which has been well developed and widely applied in various industries, software reliability is still a relatively new field, and it is generally more difficult to ensure (Xie, 1991). Also, the rapid increase of software size and complexity imposes many challenges to achieve high reliability of software products.

1.1 Background

As a matter of fact, software has become the major source of reported outages, and billions of dollars has been wasted each year (Lyu, 1996). The following are some famous examples in recent years (Charette, 2005): in 2001, software problems with supply-chain management system contributed to $100 million (USD) loss to the Nike

Inc.; in 2002, McDonald's Corp. canceled the Innovate information-purchasing system after $170 million (USD) was spent; in 2004, Hewlett-Packard Co. lost $160 million (USD) due to the software problems with ERP system and Ford Motor Co. suffered a loss of approximately $400 million (USD) deployment cost from abandoning the purchasing system. It is therefore not surprising that software reliability engineering (SRE) has received lots of attention, and abundant research has been carried out recently.

To ensure the reliability of software, software needs to be tested prior to its release. This testing phase is time-consuming and costly. During this phase, the latent software faults are identified, isolated and removed. As a result, software reliability is improved. Based on the failure data obtained from the testing phase, software reliability can be measured and predicted with appropriate software reliability models (SRMs) (Musa et al., 1987; Xie, 1991; Lyu, 1996; Pham, 2000).

The mainstream of software reliability modeling can be classified into two categories: the analytical approach and the data-driven approach (Hu et al., 2007). Analytical software reliability models (ASRMs) are generally based on certain prior assumptions made on the nature of software faults and the stochastic behavior of software failure process. These assumptions include equal fault sizes, perfect debugging, immediate fault repair, independent software failures, etc. Although these assumptions may not be valid in practice, they are made to facilitate software reliability modeling (Musa et al., 1987; Xie, 1991; Lyu, 1996; Pham, 2000).

As to the data-driven approach to software reliability modeling, the software failure process is viewed as a time series. Data-driven software reliability models (DDSRMs) are constructed to recognize the inherent patterns of the process which are carried by the recorded failure data. By modeling and analyzing the inherent patterns of software failure process, software reliability prediction can be made (Hu et al., 2007).

1.2 Motivation

SRMs are successfully applied in many real world projects, and there are more and more companies adopt the knowledge in software reliability engineering in practice (Wood, 1996; Musa, 2006). However, for both ASRMs and DDSRMs, there are still some assumptions that can be relaxed to better describe the software failure process. In addition, constructing models is not the end, to guide management when to release software is a typical application of these models. For this software release time determination problem, it is still an open question on how to describe management's attitude more accurately. Due to these considerations, research in this thesis is conducted by investigating the following specific topics.

1.2.1 Reliability Analysis for Open Source Software

Recently, a new style of software development process, the open source software (OSS) movement has received intensive interests (Raymond, 2001). OSS process is a relatively new way of building and deploying large software systems on a global basis, and differs in many interesting ways from the principles and practices of traditional software engineering (Feller et al., 2005). There is widespread recognition that open source projects can produce high quality and sustainable software systems (such as Linux operating system, Apache web server, and Mozilla browser) that can be used by thousands to millions of end-users (Mockus et al., 2002). Currently, most OSS system is developed and maintained by non-commercial communities. However, more and more software companies have switched from a closed source to an open source development model in order to win market share and to improve product quality (Hertel et al., 2003).

Since OSS is usually developed outside companies – mostly by volunteers – and the development method is quite different from the standard methods applied in commercial software development, the quality and reliability of the code needs to be investigated (Gyimothy et al., 2005). However, most existing research works have been focusing on the study of fault-proneness detection and defect prediction of OSS, which are essentially indirect reliability measurements without consideration of time effect. In fact, only in some recent studies by Tamura and Yamada (2008; 2009), such issue is considered. However, in their work, the differences between traditional commercial software and OSS are not highlighted. This motivates us to further investigate this problem by incorporating special properties of OSS into the analysis.

1.2.2 Relationship of Software Failures

Existing research on data-driven approach to software/system reliability modeling and prediction generally assumes that a failure is strongly correlated with the most recent several failures; thus the sliding window technique has been adopted to describe this relationship. However, this assumption restricts the general time series analysis to a special case as the correlation may be quite complicated in a time series (Tsay, 2002). In fact, it is possible that a failure is correlated with *some* of previous failures, not necessarily being the *most recent* ones. For example, a failure, x_i, could be correlated with, say, x_{i-8}, x_{i-6}, and x_{i-2}. If this is the case, these three time lag terms should be used as model inputs instead of using x_{i-3}, x_{i-2}, and x_{i-1}. Obviously, there should be a systematic way to discover the correlation among failures, which enables the model user to decide appropriate time lag terms to be used in the model, and hence the model performance can be improved.

1.2.3 Software Release Policy under Parameter Uncertainty

Software release time determination problem is of great importance in software development. Most existing research on this problem has been based on the assumption that parameters of software reliability models are known or accurately estimated. However, these model parameters are unknown in nature. They are generally estimated based on the limited amount of recorded failure data. Hence, the accuracy of the optimum release time obtained is questionable. It is necessary for management to know what the significant parameters are, and sensitivity analysis is needed.

In fact, the problem of parameter uncertainty has been widely discussed in many domains. Benke et al. (2008) studied the effect of parameter uncertainty on the output in a water-balance hydrological model. Yu and Harris (2009) classified the inputs into two categories and discussed this problem in the framework of global sensitivity analysis. Also, the so called robust optimization which considers the uncertainty of parameters has been received a lot of research attention recently (Ben-Tal and Nemirovski, 2002; Sahinidis, 2004). Previous research has demonstrated that parameter uncertainty cannot be discarded in the modeling and analysis. This also motivates us to study the optimal software release policy under parameter uncertainty.

1.2.4 Formulation of Software Release Time Determination Problem

For software release time determination problem, reliability and cost are two important dimensions that are generally considered. In order to determine an optimal software release time, existing research formulates this problem in the following three ways: (1) cost minimization (Boland and Singh, 2003; Morali and Soyer, 2003; Xie and Yang, 2003; Huang and Lyu, 2005a), (2) cost minimization given a reliability constraint (Yamada and Osaki, 1985; Pham, 1996; Pham and Zhang, 1999; Huang, 2005; Boland and Chuiv, 2007), and (3) reliability maximization under a cost budget (Leung, 1992). It can be seen that software release time determination problem is formulated as single-objective optimization problems. However, this kind of formulation can hardly describe the management’s attitude accurately. In reality, maximizing reliability and minimizing cost is expected to be considered simultaneously, and a compromise should be made between these two objectives based on management’s preference. This motivates us to develop a new formulation for software release time determination problem, such that a more reasonable decision can be made.

1.3 Objective and Scope of Research

The objective of this thesis is to develop comprehensive and practical models for software reliability analysis and software release time determination. Both ASRMs and DDSRMs are extended considering the practical issues involved in software reliability modeling. More specifically, in the framework of ASRMs, a model for open source software (OSS) is developed by incorporating the special properties of OSS; in the framework of DDSRMs, a generic model is proposed by relaxing the basic assumption in most existing DDSRMs.

Besides the modeling part of software failure process, software release time determination problem, as a typical application of SRMs, is investigated as well. Sensitivity analysis of release time is introduced as a way to deal with parameter uncertainty. In particular, sensitivity of the software release time is investigated through various methods, including one-factor-at-a-time approach, design of experiments and global sensitivity analysis. By comparing different approaches, applicability and limitations of them will be shown.

However, sensitivity analysis can only identify significant parameters. It is still imperative to investigate the release policy under parameter uncertainty. Theoretically, it can be shown that there is about 50% risk that software reliability requirement cannot be met when the mean value is used. This is because model parameters are unknown in nature, and they are estimated based on the limited amount of data. Provided that the 50% risk can be too high to be acceptable for management, software release policy under parameter uncertainty is studied.

Furthermore, for release time determination problem, most existing research formulates it as single-objective optimization problems, which can hardly describe the decision process accurately. Therefore, multi-objective optimization models are developed for software release time determination problem, and different multi-objective optimization approaches are adopted for analysis.

The remainder of this thesis is organized as follows. Chapter 2 provides a general review on software reliability modeling and the corresponding release time determination problem. In Chapter 3, reliability analysis and optimal version-updating for open source software is studied. Chapter 4 discusses the proposed generic data-drive software reliability model with model mining technique. Chapter 5 discusses the sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. Chapter 6 highlights the risk that software cannot meet its reliability requirement due to parameter uncertainty. Also, a risk-based approach for release time determination with delay costs considerations is introduced. Chapter 7 formulates the software release time determination problem as multi-objective optimization problems, and different multi-objective optimization approaches are compared. Chapter 8 concludes current research works and looks at future research prospects.

Chapter 2 Literature Review

Software reliability modeling is of great importance for the reason that it can measure the reliability of software quantitatively by analyzing the recorded failure data. Due to this, a large number of software reliability models have been proposed and published in the literature (Musa et al., 1987; Xie, 1991; Lyu, 1996; Pham, 2000). In this chapter, a brief review on software reliability modeling is given, focusing on the two general categories: analytical software reliability models (ASRMs) and data-driven software reliability models (DDSRMs) (Hu et al., 2007). In addition, software release time determination, as a typical application of software reliability models, is briefly reviewed as well.

2.1 Analytical Software Reliability Models

Analytical software reliability models (ASRMs) are generally based on certain prior probabilistic assumptions made on the stochastic behavior of software failure process, such as the Markov process assumption and non-homogenous Poisson process (NHPP) assumption. It is worth noting that most of the Markov models are times-between-failures models and almost all NHPP models are failure-count models according to the classification system proposed by Goel (1985). In the following sub-sections, the Jelinski-Moranda model and a general formulation of NHPP models will be briefly introduced. In addition, some recent advances on ASRMs will be discussed as well.

2.1.1 The Jelinski-Moranda Model

From a historic point of view, the Jelinski-Moranda model (Jelinski and Moranda, 1972) has a paramount influence on software reliability modeling. It is the first published Markov model and the main assumptions of this model are:

(1) The number of initial software faults is an unknown but fixed constant.
(2) A detected fault is removed immediately and no new faults are introduced.
(3) Times between failures are independent, exponentially distributed random quantities.
(4) Each remaining software fault contributes the same amount to the software failure intensity.

In the Jelinski-Moranda model, let N_0 denote the number of initial faults in the software before the testing starts; the initial failure intensity is then $N_0\phi$, where ϕ is a constant denoting the failure intensity contributed by each fault. Let T_i , $i = 1,2,\cdots,N_0$ denote the time between (*i*-1)th and *i*th failures, then T_i 's are independent, exponentially distributed random variables with parameter

$$\lambda_i = \phi[N_0 - (i-1)], \quad i = 1,2,\cdots,N_0 \tag{2.1}$$

It is obvious that the failure intensity is constant between the detection of two consecutive failures. This is quite reasonable since the software is unchanged between the detection of two consecutive failures and the testing process is random and homogeneous. However, the assumption that software faults are of the same size contributing the same amount to software failure intensity is not realistic. In order to relax this unrealistic assumption, some extensions of Jelinski-Moranda model were made, see, e.g., Schick and Wolverton (1978), Shanthikumar (1981), Xie (1990), Chang and Liu (2009). However, due to the complexity of these models, they have not been widely applied in practice compared with the NHPP models, which will be discussed in the following section.

2.1.2 A General Formulation of NHPP Models

NHPP models form a major part of analytical software reliability modeling. In these models, the underlying software fault detection process is assumed to be a non-homogeneous Poisson process. As software faults are detected, isolated and removed, the software being tested becomes more reliable with a decreasing failure intensity function. In general, an NHPP software reliability growth model (SRGM) can be developed by solving the following differential equation (Pham, 2003):

$$\frac{dm(t)}{dt} = b(t)\left[a(t) - m(t)\right], \tag{2.2}$$

where *m*(*t*) is the mean value function of detected faults, *a*(*t*) and *b*(*t*) are fault content function and failure detection rate function respectively. It can be seen that the idea behind the above equation just stems from the Jelinski-Moranda model, where the relation between failure intensity and the number of remaining faults is studied. Given different expressions and explanations of $a(t)$ and $b(t)$, different NHPP SRGMs can be obtained (Zhang and Pham, 2000).

Specifically, when $a(t) = a$, $b(t) = b$, the important Goel-Okumoto (GO) model is received (Goel and Okumoto, 1979). This model has strongly influenced the development of many later models. Actually, many later NHPP models are modifications or generalizations of this model. It should be noted that in the Goel-Okumoto model, both two model parameters are positive, and there are some physical meanings of them. In particular, *a* represents the number of faults to be eventually detected, and *b* denotes the failure detection rate per fault.

The Goel-Okumoto model was successfully applied in many projects as reported in Wood (1996). However, it was sometimes observed that the curve of the cumulative number of faults is S-shaped. The reason for the S-shaped behavior is the “learning” effect of the debugging process (Yamada et al., 1984). To consider this issue, the delayed S-shaped NHPP model (Yamada et al., 1983; 1984) and the inflected S-shaped NHPP model (Ohba, 1984) were proposed. For these two models, we still have $a(t) = a$. The only difference is the failure detection rate function. Specifically, $b(t) = b^2 t \big/ (1 + bt)$ is in the delayed S-shaped model (Yamada et al., 1983; 1984) and $b(t) = b \big/ (1 + ce^{-bt})$ is in the inflected S-shaped NHPP model (Ohba, 1984).

2.1.3 Recent Advances on ASRMs

Based on the above discussions, it can be seen that different assumptions indicate different descriptions of the software failure process. However, it is worth noting that the underlying software failure process can hardly be described precisely, and assumptions are made to develop a model for the sake of mathematical tractability (Goel, 1985). It is obvious that these assumptions are in most cases not valid and cannot be made in some practical applications. Thus, relaxing these assumptions has drawn a lot of research attention, and most recent advances on ASRMs were focused on a better description of the software failure process and more reasonable software reliability analysis.

Fault Correction Process

Most existing SRGMs assume that faults are immediately removed when failures are detected, i.e., the repair time is ignored. Although this assumption provides simplicity and mathematical tractability for the modeling of the software failure process, it is usually not the case. In reality, the fault removal activity rarely occurs immediately after the observation of failure, and the time needed to correct the fault cannot be ignored.

Schneidewind (1975) first modeled the fault correction process following the fault detection process with a constant time delay. However, a constant time delay assumption may not be appropriate since faults cannot be corrected with the same amount of testing effort in reality. For example, based on the empirical study of nearly

200 anomalies from seven NASA spacecraft systems, it was found that some anomalies are in need of multiple corrections (Lutz and Mikulski, 2004). With the consideration of this, some extensions are made under the framework of the model proposed by Schneidewind (1975). Xie and Zhao (1992) substituted the constant time delay with a time dependent delay function in their model, with the assumption that detected faults become harder to be corrected as the testing proceeds. Schneidewind (2001) assumed that the time delay is an exponentially distributed random variable. Recently, Xie et al. (2007) carried out a comprehensive study of the time delay issues with different kinds of distributions. Wu et al. (2007) discussed the parameter estimations of the combined model. Moreover, Huang and Lin (2006) incorporated both fault dependency and debugging time lag into the modeling.

The models discussed above incorporated the correction process into analysis by introducing a time delay function. In fact, there also exist other alternative ways. More specifically, Bustamante and Bustamante (2002) proposed a software reliability model which represents the software failure process with a non-homogeneous Poisson Process and the correction process with a multinomial distribution. Gokhale et al. (2004; 2006) modeled both fault detection process and correction process with a non-homogeneous Markov chain, where different fault removal policies are studied by different forms of the fault removal rate. Lo and Huang (2006) proposed a general framework for modeling these two processes with assumption that the mean number of faults corrected in a very small time interval is proportional to the mean number of detected but not yet corrected faults remaining in the system. Huang and Huang (2008) introduced the use of finite and infinite server queueing models to describe these two processes, and the correction process is studied by cumulative distribution function of failure correction time.

However, most of the existing models considering both of these processes assume that the failure rate at current time is proportional to the number of remaining undetected faults. In fact, since the fault removal activity is considered, this assumption no longer holds, and it is more reasonable to assume that the failure rate at time *t* is proportional to the number of remaining uncorrected faults (Hwang and Pham, 2009).

Incorporation of Testing Effort

In recent years, incorporating testing effort into software reliability growth models (SRGMs) has received a lot of attention, probably because testing effort is an essential process parameter for management. Huang et al. (2007) showed that logistic testing effort function can be directly incorporated into both exponential-type and S-type non-homogeneous Poisson process (NHPP) models, and the proposed models were also discussed under both ideal and imperfect debugging situations. Kapur et al. (2007) discussed the optimization problem of allocating testing resources by using marginal testing effort function (MTEF). Later, Kapur et al. (2008a) studied the testing effort dependent learning process, and faults were classified into two types by the amount of testing effort needed to remove them. In addition, some research incorporated change-point analysis in their models as the testing effort consumption may not be smooth over time (Huang, 2005; Kapur et al., 2008b; Lin and Huang, 2008). Moreover, as constructing model is not the end, the optimal release time problem considering testing effort was also discussed (Yamada et al., 1993; Huang and Kuo, 2002; Huang and Lyu, 2005a; Lin and Huang, 2008).

However, most of the research assumes that parameters of the proposed models are known. In fact, there always exist estimation errors as parameters in testing effort function and SRGMs are generally estimated by least squares estimation (LSE) and maximum likelihood estimation (MLE) methods respectively. It is necessary to conduct the sensitivity analysis to determine which parameter may have significant influence to the software release time. This is even more important when there are an increasing number of parameters involved in the model.

2.2 Data-Driven Software Reliability Models

In data-driven approach, the software failure process is viewed as a time series, and data-driven software reliability models (DDSRMs) are constructed to recognize the inherent patterns of the process which are carried by the recorded failure data. By modeling and analyzing the inherent patterns of software failure process, software reliability prediction can be made. The main advantage of DDSRMs is that they do not require restrictive assumptions on software faults or software failure process; thus they may have better applicability across different software projects compared with traditional ASRMs.

Machine learning techniques like artificial neural networks (ANNs) and support vector machines (SVMs) have been successfully applied for constructing DDSRMs.

For ANNs, both feed-forward neural networks and recurrent neural networks were used and compared in software reliability analysis (Karunanithis et al., 1992; Sitte, 1999). Later, Cai et al. (2001) investigated the effectiveness of the use of ANNs in software reliability prediction, and found that ANNs' performance is highly dependent on the 'smooth' trend of the data. Ho et al. (2003) revisited the connectionist model with a modified Elman recurrent neural network, which outperforms both of the Jordan model and feed-forward model. Tian and Noore (2005a) used genetic algorithm (GA) to optimize the number of the delayed input neurons and the number of neurons in the hidden layer of the neural network architecture. Su and Huang (2007) developed a dynamic weighted combinational model for software reliability prediction, and the results showed that the proposed model has a fairly accurate prediction capability. Hu et al., (2007) applied recurrent neural networks to model both the fault detection process and the fault correction process in software testing, and the authors proposed a GA-based networks configuration approach.

Besides ANNs, another machine learning technique that has emerged as a promising modeling paradigm is support vector machines (SVMs), which have good generalization capability due to the structural risk minimization principle used (Vapnik, 1995; Vapnik, 1999; Kecman, 2001; Scholkopf and Smola, 2002). SVMs have been successfully applied in many domains such as pattern recognition, time series forecasting, diagnostics, robotics, and process control. In software reliability modeling and prediction domain, SVM-based SRMs have been proposed and studied as well. Tian and Noore (2005b) proposed an SVM-based modeling approach to software reliability prediction, and experimental results showed that the proposed approach adapts well across different software projects and has higher next-step prediction accuracy compared with feed-forward ANN and recurrent ANN modeling approaches. Pai and Hong (2006) proposed an SVM-based SRM which uses simulated annealing (SA) algorithms to optimize model parameters.

However, most existing research on data-driven approach to software reliability modeling and prediction generally assumes that a failure is strongly correlated with the most recent several failures. This assumption restricts the general time series analysis to a special case as the correlation may be quite complicated in a time series (Tsay, 2002). There should be a systematic way to discover the correlation among failures, which enables the model user to decide appropriate time lag terms to be used in the model, and hence the model performance could be improved.

2.3 Determination of Software Release Time

Constructing software reliability models is not the end. It is almost always the case that the model is developed to help management make some decisions. A typical purpose is to guide management on when to release/sell the software in the market. Since Okumoto and Goel (1980) firstly proposed the determination of software release time problem in 1980, many research works have been done in the past several decades.

Koch and Kubat (1983) introduced the penalty cost into the release time determination model. Yamada and Osaki (1985) proposed a decision-making model, where both reliability and cost are considered. In particular, their model was developed to minimize the cost subject to a reliability constraint. Dohi (1999) transformed the optimal software release time problem into a time series prediction problem, and the artificial neural network (ANN) was employed. Nishio and Dohi (2003) presented the determination of the optimal software release time based on proportional hazards software reliability growth model. Huang and Lyu (2005) proposed the optimal release time policy for software systems considering cost, testing-effort, and test efficiency, which enriched the decision model. Xie and Yang (2003) and Boland and Chuiv (2007) considered the optimal software release time when repair is imperfect. Chiu (2009) proposed a Bayesian method to determine the optimal release time for software systems based on experts' prior judgments.

It is worth noting that the uncertainty involved in the determination of optimal release time has received special attention recently (Yang et al., 2008; Ho et al., 2008). It has been pointed out that the point estimate received from the traditional way is not precise as the software debugging process is essentially random. Yang et al. (2008) introduced a risk-control approach to obtain the optimal release time by quantifying the uncertainty in the actual cost of the project by variance. Ho et al. (2008) determine the optimal release time by considering the randomness of the mean value function, and the randomness is assumed to stem from the error-detection process. However, the optimal release policy considering the parameter uncertainty is still lacking.

Furthermore, for the determination of optimal software release time, reliability and cost are the two important dimensions that are generally considered. It should be noted that most existing research formulates this decision process as single-objective optimization problems. Although these formulations can greatly reduce the complexity, they can hardly reflect the nature of the decision process, which is essentially a multi-objective optimization problem. More specifically, maximizing reliability and minimizing cost should be achieved simultaneously.

Chapter 3 Reliability Analysis and Optimal Version-Updating for Open Source Software

3.1 Basic Problem Description

Open source software (OSS) development is a new way of building and deploying large software systems on a global basis, and it differs in many interesting ways from the principles and practices of traditional software engineering (Raymond, 2001). There is a widespread recognition across software industry that open source projects can produce software systems of high quality and functionality, such as Linux operating system, Apache web server, Mozilla browser, MySQL database system, etc., that can be used by thousands to millions of end-users (Mockus et al., 2002).

The OSS development is based on a relatively simple idea: the original core of the OSS system is developed locally by a single programmer or a team of programmers. Then a prototype system is released on the internet, so that other programmers can freely read, modify and redistribute that system's source code. The evolution process of OSS is much faster than the closed source project. The reason is that in the development of OSS, tasks are completed without assigning from hierarchical management and there is no explicit system-level design, no well-defined plan or schedules. A central managing group may check the code but this process is much less rigid than in closed-source projects.

Several OSS systems have been in widespread use with thousands or millions of end-users, e.g. Mozilla, Apache, OpenOffice, Eclipse, NetBeans, GNOME, and Linux. Due to the success of OSS, more and more software companies have switched from a closed source to an open source development in order to win market share and to improve product quality (Ven and Mannaert, 2008). Even the leading commercial software companies, such as IBM and Sun, have begun to embrace the open source model and are actively taking part in the development of OSS products.

As OSS application rapidly spreads out, it is of great importance to assess the reliability of OSS system to prevent potential financial loss or reputational damage to the company (Gyimothy et al., 2005). Due to this consideration, many studies have been carried out recently on predicting number of defects in the system. For instances, Eaddy et al. (2008) investigated the relationship between the degree of scattering and the number of defects by stepwise regression and other statistical techniques. Marcus (2008) proposed a new measure named Conceptual Cohesion of Classes (C3) to measure the cohesion in object-oriented software. They also applied C3 in logistic regression to predict software faults with the comparisons with other object-oriented metrics. Kim et al. (2008) introduced a new technique for predicting latent software bugs in OSS, called change classification. Change classification uses a machine learning classifier to determine whether a new software change is more similar to prior buggy changes or clean changes. In this manner, change classification predicts the existence of bugs in software changes.

Although the works above can provide important information to assess the reliability for OSS, the total number of defects in a software system is an essentially indirect reliability measurement where the time factor is often neglected (Xie, 1991). Only in some recent studies by Tamura and Yamada (2008; 2009), such issue is considered. In particular, Tamura and Yamada (2008) combined neural network and software reliability growth modeling for the assessment of OSS reliability. In Tamura and Yamada (2009), the stochastic differential equation is introduced for the modeling of OSS reliability, and optimal version-update time is discussed based on it.

In this chapter, we will further investigate the modeling of OSS reliability and its optimal version-update time determination. Our model is based on non-homogeneous Poisson process (NHPP) which has been proven to be a successful model for software reliability (Musa, 1987; Xie, 1991; Lyu, 1996; Pham, 2000). However, different from the NHPP models for closed source software and the models proposed in Tamura and Yamada (2008; 2009), our model incorporates the unique patterns of OSS development, such as the multiple releases property and the hump-shaped fault detection rate function. In addition, because the project cost is no longer a crucial factor for optimal release time determination for most OSS projects, in this study, we formulate a new version-update time determination problem for OSS. Specifically, the multi-attribute utility theory (MAUT) is adopted for this decision process, where two important strategies are considered simultaneously: rapid release of the software to maintain sufficient volunteers involved and the acceptable level of OSS reliability.

The rest of this chapter is organized as follows. Section 3.2 describes our proposed model based on NHPP incorporating unique properties of OSS. Section 3.3 formulates the optimal version-update time problem based on MAUT, where the rapid release strategy and the level of reliability are considered simultaneously. Section 3.4

provides numerical examples for validation purpose based on the real world data sets. Conclusions are made in Section 3.5.

3.2 Modeling Fault Detection Process of Open Source Software

The underlying software fault detection process is commonly assumed to be a non-homogeneous Poisson process (NHPP) (Musa, 1987; Xie, 1991; Lyu, 1996; Pham, 2000). As software faults are detected, isolated and removed, the software being tested becomes more reliable with a decreasing failure intensity function. In general, an NHPP software reliability growth model (SRGM) can be developed by solving the following differential equation (Pham, 2003):

$$\frac{dm(t)}{dt} = b(t)\left[a(t) - m(t)\right], \tag{3.1}$$

where $m(t)$, $a(t)$ and $b(t)$ are the mean value function of detected faults, the fault content function and fault detection rate function respectively, and typical boundary point is $m(0) = 0$. Given different expressions and explanations of $a(t)$ and $b(t)$, many NHPP SRGMs can be developed (Zhang and Pham, 2000).

The basic assumption illustrated by the above formulation can also hold in the context of OSS (Tamura and Yamada, 2009). The reason lies in the fact that in OSS the failure rate at current time is still determined by the product of the fault detection rate function and the number of remaining faults. However, it is worth noting that some special properties of OSS have not been considered in traditional NHPP SRGMs.

One special property of OSS is that multiple releases are common and often (Kozlov et al., 2008). Hence, a general NHPP software reliability model for OSS can be developed based on the following equation

$$\frac{dm_i(t)}{dt} = b_i(t)\left[a_i(t) - m_i(t)\right] \tag{3.2}$$

where $m_i(t)$, $a_i(t)$ and $b_i(t)$ are the mean value function of detected faults, the fault content function and failure detection rate function for release *i* respectively. As to the time basis, *t* starts from zero for each new release of OSS. It should be noted that we treat each new release as a new version of software since the defect count of a previous release and its current release do not correlate with each other in most projects (Illes-Seifert and Paech, 2010). Although the previous release has some uncorrected faults which may still exist in the new release, these faults will be counted again in the system if they are found. Therefore, $m_i(0) = 0$ for each release.

Besides the multiple releases property, the fault detection process in the development of OSS is essentially different from that of traditional closed source software. The testing process of traditional closed source software relies on a specified testing team, where the number of testers is generally stable. Therefore, the constant fault detection rate has become a common assumption, such as in the famous Goel-Okumoto (GO) model (Goel and Okumoto, 1979). Moreover, to account for the "learning" effects of the testing team, the increasing fault detection rate function is used, and these models are S-shaped models as discussed in Yamada et al. (1983), Ohba (1984).

Unlike traditional closed source software, OSS involves much more testers in the testing process, and most of these testers are volunteers. The number of volunteers involved in the OSS is largely influenced by the attractiveness of the software (Raymond, 2001). More specifically, each release of OSS can attract increasing number of volunteers in the early phase since more and more people know it and use it. After the number of volunteers reaches at the peak, it will decrease since the software is losing its attractiveness over time. Accordingly, it is reasonable to assume that the fault detection rate in OSS follows a hump-shaped curve. In order to describe this special property in OSS, the first derivative of logistic function is selected, and it is given by

$$b_i(t) = \frac{N_i A_i \alpha_i \exp\left(-\alpha_i t\right)}{\left[1 + A_i \exp\left(-\alpha_i t\right)\right]^2} \tag{3.3}$$

where N_i is the product of total amount of testing effort eventually consumed and a constant fault detection rate (Huang and Kuo, 2002; Huang et al. 2007), A_i scales the $b_i(t)$ without changing its shape and α_i is the shape parameter of $b_i(t)$ for each release i. It is worth noting here that $b_i(t)$ reaches its maximum value $b_{i\max} = N_i \alpha_i / 4$ at

$$t_{i\max} = \frac{1}{\alpha_i} \ln A_i \, . \tag{3.4}$$

Since the fluctuation in $b_i(t)$ originates from the change in the number of volunteers involved in the fault detection process, $t_{i\max}$ indicates that the testing effort from volunteers reaches its maximum at this time for release *i*.

Moreover, the Weibull-type fault detection rate function can also capture the property of the hump-shaped curve with fairly good flexibility. That is

$$b_i(t) = N_i \beta_i \gamma_i t^{\gamma_i - 1} \exp\left(-\beta_i t^{\gamma_i}\right) \tag{3.5}$$

where N_i is still the product of total amount of testing effort eventually consumed and a constant fault detection rate (Huang and Kuo, 2002; Huang et al. 2007); β_i and γ_i are the scale parameter and shape parameter respectively for each release *i*. However, the use of Weibull function suffers from two major deficiencies which may restrict their applicability in OSS reliability modeling. First, when $\gamma_i = 1$, $b_i(t)$ is a monotonic decreasing function over time. This cannot capture the special property of OSS where the hump-shaped curve is the case. Second, when $\gamma_i \neq 1$, $b_i(0)$ is always equal to zero and this actually introduces a bias into the modeling. Specifically, each version of the OSS has a number of volunteers (if no volunteers, at least developers) at the starting time. Therefore, $b_i(0)$ should be a non-zero value.

The selection of the logistic function can overcome the disadvantages of the Weibull function. Not only does it have good flexibility to describe the hump-shaped curve, it can also provide a more reasonable starting point with a non-zero value. In Huang and Kuo (2002) and Huang et al. (2007), the differences between the use of Weibull-type function and logistic function in software reliability modeling were also discussed, and interested readers could refer to them for more detailed discussions.

3.3 Determination of Optimal Version-Update Time

Optimal release time determination in the testing phase is a typical application of software reliability models. The total expected cost including both testing cost and operation cost is a crucial factor for such determination (Pham, 2003). However, most OSS projects are interest-driven, and most development activities in OSS projects are accomplished by volunteer users. Consequently, the cost is no longer an important consideration for the OSS community to decide the release time (Samoladas et al., 2010).

For OSS development, there are two important factors for management to determine the optimal version-update time in the testing phase. On one hand, a sufficient number of volunteers are expected to be involved in the development of OSS. Since volunteers are interest-driven and the attractiveness of OSS is of great importance for them, rapid release of OSS becomes critical for maintaining the number of current volunteers and attracting new comers (Raymond, 2001). On the other hand, reliability, as the most important aspect of OSS quality, has to be ensured as well (Tamura and Yamada, 2009). Since reliability is an increasing function over time, reliable software requires a delay of the release to ensure that there is sufficient time for testing.

One challenging issue is that the rapid release strategy and the level of reliability are essentially conflicting with each other. Management, therefore, has to make a compromise between them. To the best of our knowledge, discussions on such a problem are still lacking in the literature, which motivates us to develop a new decision model. To tackle these two conflicting factors simultaneously, multi-attribute utility theory (MAUT) is adopted in our decision model.

In MAUT, some independence assumptions, such as preferential independence, utility independence and additive independence, are used for a more practical form of the multi-utility function. It is worth noting that these assumptions are commonly accepted in practice. Moreover, it has been shown that even when these assumptions are violated, the additive multi-attribute function can provide fairly good approximations (Edwards, 1977; Farmer, 1987). For more detailed discussions on the multi-attribute function when independence assumptions are not held, interested readers can refer to (Keeney and Raiffa, 1976). In this thesis, we will adopt these commonly used assumptions.

The application of MAUT can obtain a one-dimensional multi-attribute utility function, which is the measure of the attractiveness of the conjoint outcome of attributes given a specified alternative. The additive form of the multi-attribute utility function is given by

$$U(d_1, d_2, \ldots d_n) = \sum_{i=1}^{n} w_i u(d_i) \tag{3.6}$$

where each attribute is denoted by d_i, i=1,2,…n, the attractiveness of each attribute is represented by the single utility function $u(d_i)$ and w_i's are the scaling constants allocated for different single utility functions. The scaling constants represent the different importance weights for the utilities of attributes and their sum is equal to one (von Winterfeldt and Edwards, 1986). By maximizing the multi-attribute utility function, the best alternative is obtained, under which the attractiveness of the conjoint outcome of attributes is optimized.

The main reason for the selection of MAUT in our problem is that scenarios of management can be appropriately represented by the structure of it. Furthermore, MAUT has strong theoretical foundations based on the expected utility theory (Fishburn, 1970). Last but not least, as indicated in Ferreira et al. (2009), the use of MAUT provides the feasibility to consider the alternative on the continuous scale. The procedure of the use of it in our problem is discussed in detail as follows.

3.3.1 Quantification of Attributes

One strategy to the success of open source software is the rapid release of the software. Such a strategy can ensure a sufficient number of volunteers involved in the testing process. However, the real number of volunteers and their testing effort can hardly be traced and measured over time. To resolve this difficulty, analyzing the failure data available for the determination of underlying volunteers' testing effort could be an alternative. Fortunately, the proposed model in this chapter possesses such an advantage because the fault detection rate function $b_i(t)$ can describe the underlying change of volunteers' testing effort by the logistic function. Therefore, the objective of rapid release can be formulated as.

$$\text{Maximize } \beta_i = b_i(t)/b_{i\max} \tag{3.7}$$

where the rapid release indicator β_i for release *i* is one of the attributes to be considered in MAUT, and *t* is the decision variable, $\beta_i \in (0,1]$. In particular, a large value of it indicates a rapid release, and it reaches its maximum at the time $t_{i\max}$.

On the other hand, during the testing process of OSS, maximizing software reliability is also a major concern of management. A simple index to measure the reliability is the ratio of the number of cumulative detected faults at time *t* to the mean value of initial faults in the software (Lin and Huang, 2008). Hence, the reliability for release *i* can be represented by $m_i(t)/a_i$, and it should be maximized.

$$\text{Maximize } R_i = m_i(t)/a_i \tag{3.8}$$

where the approximated reliability R_i is another attribute in MAUT, and *t* is the decision variable. Since the reliability of release *i* is an increasing function of time, it reaches its maximum when time goes to infinity. Therefore, when both rapid release indicator and reliability are considered, the decision space is $[t_{i\max}, +\infty)$, and $R_i \in [m_i(t_{i\max})/a_i, 1)$.

3.3.2 Elicitation of Single Utility Function for Each Attribute

The single utility function for each attribute represents management's satisfaction level towards the performance of each attribute. It is usually assessed by a few particular points on the utility curve (Keeney and Raiffa, 1976; von Winterfeldt and Edwards, 1986). More specifically, suppose that the single utility function for reliability is to be determined, the worst and best values of reliability are selected first as R_i^0 and R_i^1. For OSS management, these values are of great importance because R_i^0 and R_i^1 represent its lowest reliability requirement and its highest reliability expectation respectively. At these boundary points, we have $u\left(R_i^0\right)=0$ and $u\left(R_i^1\right)=1$. Here, the subscript and the superscript of R_i^j represent the number of release and the parameter's corresponding utility value respectively and $j \in [0,1]$.

Subsequently, management is presented with some simple hypothetical gambles to determine the certainty equivalents for a few 50-50 lotteries (Keeney and Raiffa, 1976; von Winterfeldt and Edwards, 1986). For example, management is asked to chose a value for $R_i^{0.5}$, so that it is indifferent between accepting $R_i^{0.5}$ with certainty and having a 50-50 lottery, where there are 0.5 probabilities of getting R_i^0 and R_i^1 respectively. Similarly, $R_i^{0.75}$ can be determined with a 50-50 lottery which consists of $R_i^{0.5}$ and R_i^1. Also, $R_i^{0.25}$ can be obtained with a 50-50 lottery which includes R_i^0 and $R_i^{0.5}$. These five points are commonly used to elicit the single utility function for each attribute (Keeney and Raiffa, 1976), which is generally represented by the linear or exponential function as

$$u(R_i) = l + m \cdot R_i \text{ or } u(R_i) = l + m\exp(n \cdot R_i) \tag{3.9}$$

where *l*, *m* and *n* are constants which secure $u(R_i) \in [0,1]$. It should be noted that we also need to compare the certainty equivalents and the expected values of the 50-50 lotteries to determine which form in (3.9) should be selected. Specifically, if they are equal to each other, management is risk neutral and the linear form should be used. Otherwise, management is not risk neutral and the exponential form is generally adopted.

The single utility function $u(\beta_i)$ for the rapid release indicator can be obtained as well. Similarly, β_i^0 and β_i^1 are very important for management because they denote its lowest rapid release requirement and its highest rapid release expectation respectively.

3.3.3 Estimation of Scaling Constants

The following step is the estimation of the scaling constants w_{β_i} and w_{R_i}. For real applications in OSS projects, they indicate the importance weights that management allocates for each attribute (von Winterfeldt and Edwards, 1986). There are two common methods to assess the scaling constants: certainty scaling and probabilistic scaling (von Winterfeldt and Edwards, 1986). Given that the number of attributes considered in our problem is only two and this is a small number, the probabilistic scaling technique is recommended for use.

In probabilistic scaling, management is asked to compare two choices as shown in Figure 3.1. On the left hand side, there is a certain joint outcome $\left(\beta_i^1, R_i^0\right)$ comprised of rapid release indicator at its best level and reliability at its worst level. On the right hand side, the lottery is comprised of both attributes at their best levels with probability *p* and of both attributes at their worst levels with probability 1-*p*.

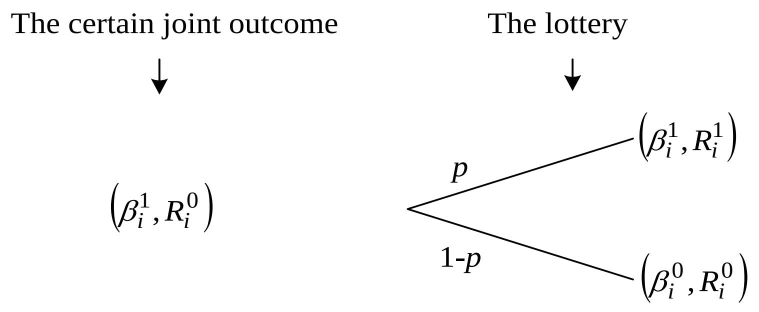

The certain joint outcome
The lottery
$\left(\beta_i^1, R_i^0\right)$
p
$1-p$
$\left(\beta_i^1, R_i^1\right)$
$\left(\beta_i^0, R_i^0\right)$

Figure 3.1 Two choices for the determination of the scaling constant w_{β_i}

In the beginning, management is asked to compare the certain outcome with the lottery having a 50-50 chance of occurring. If management prefers the certain outcome, the probability *p* is gradually increased until management is indifferent with these two choices. On the contrary, if management prefers the lottery, we decrease the probability *p* until management's indifference towards these two choices is achieved. At indifference, *p* is equal to the scaling constant w_{β_i} for the rapid release indicator. Since the sum of scaling constants must be equal to one, the other scaling constant w_{R_i} can be obtained with ease.

3.3.4 Maximization of Multi-Attribute Utility Function

Based on the previously estimated single utility functions and scaling constants, the additive form of the multi-attribute utility function in our problem can be obtained. That is

$$U(\beta_i, R_i) = w_{\beta_i} u(\beta_i) + w_{R_i} u(R_i) \tag{3.10}$$

where w_{β_i} and w_{R_i} are the scaling constants for attribute β_i and R_i respectively and $u(\beta_i)$ and $u(R_i)$ are the single utility function for each attribute. By maximizing this multi-attribute utility function, the optimal version-update time T_i^* is obtained.

It is worth noting here that the additive form of multi-attribute utility function is based on the utility independence assumption and the additive independence assumption. Interested readers can refer to Keeney and Raiffa (1976) for more detailed theoretical discussions. However, from the real applications' point of view, these assumptions are commonly accepted in practice (Brito and de Almeida, 2009; Ferreira et al., 2009).

3.3.5 Summary of the Procedure

The procedure of the use of MAUT in our problem is summarized in Figure 3.2. The first step of the implementation of the decision model is to quantify the attributes in our problem, which are the rapid release indicator and the reliability. For the rapid release indicator, it is quantified by (3.7) based on the failure data collected during the testing process. While for the attribute of reliability, it represents the ratio of cumulative detected faults at time *t* to the mean value of initial faults and it is quantified by (3.8). The following step is the elicitation of the single utility functions for both attributes. As discussed previously, the linear form and the exponential form in (3.9) are generally used. After this, the scaling constants for each attribute are estimated by comparing the two choices as shown in Figure 3.1. Finally, based on the single utility functions and the scaling constants, the multi-attribute utility function is obtained as shown in (3.10). By maximizing this multi-attribute utility function, the optimal version-update time for release *i* is determined, which is the best option of version-update time when the rapid release strategy and the reliability of software are considered simultaneously.

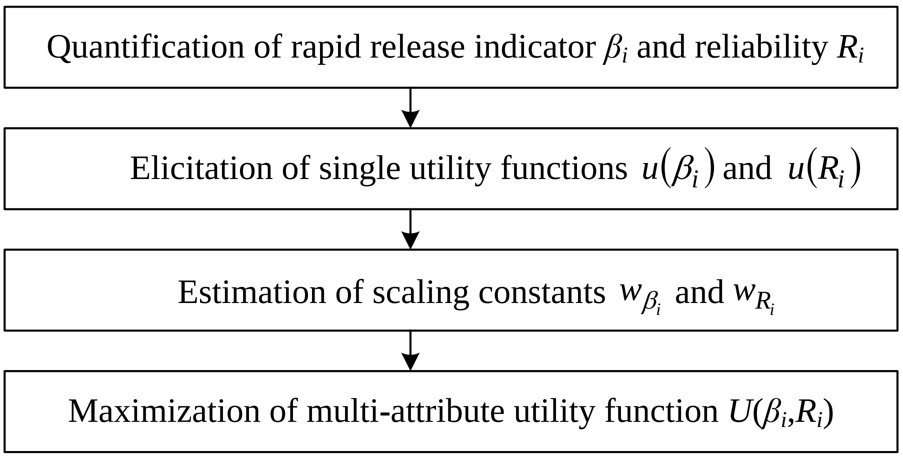

Quantification of rapid release indicator β_i and reliability R_i
Elicitation of single utility functions $u(\beta_i)$ and $u(R_i)$
Estimation of scaling constants w_{β_i} and w_{R_i}
Maximization of multi-attribute utility function $U(\beta_i,R_i)$

Figure 3.2 The structure of the decision model for the determination of optimal version-update time

3.4 Numerical Examples

Special properties of OSS are incorporated into the proposed model for open source software reliability. In order to compare the proposed model against traditional models for reliability assessment, numerical examples are provided based on two real world data sets from two famous open source projects: Apache and GNOME. Furthermore, based on the failure data from the first release of Apache, a decision model application example is provided, and sensitivity analysis is introduced to help management know how robust the decision is.

3.4.1 The Data Sets

Enormous open source projects are undergoing development and each project generates a lot of data sets. Therefore, it is important to select representative open source projects for model validations. Apache and GNOME projects both have large and well-organized communities, where a great number of developers have the right to update and change files freely. The large sizes of these two projects make them the state-of-the-art in terms of management of OSS projects.

Apache 2.0.35 is available to the public since 2002/04/06 and this is the first release of Apache's major version 2.0. We select this release and the following two as our examples. As to the GNOME project, GNOME 2.0 is a major upgrade which includes the introduction of the human interface guidelines. Hence, this release and the following two releases are adopted for our test beds. The retrieved faults are presented in Table 3.1 and Table 3.2 respectively. In these two tables, some failure data is not shown for simplicity. For example, since there are no faults detected on the 29th day in Apache 2.0.35, failure data on this day is not shown in Table 3.1.

Table 3.1 Detected faults in Apache official public releases

Apache 2.0.35		Apache 2.0.36		Apache 2.0.39	
Days from release	Detected bugs	Days from release	Detected bugs	Days from release	Detected bugs
1	5	1	2	1	1
2	5	2	5	2	2
3	4	3	1	3	2
4	1	4	1	4	3
5	2	5	1	5	3
6	4	7	2	7	2
7	6	8	1	8	1
8	2	9	1	9	1
10	1	10	3	10	1
11	8	12	2	11	1
12	5	13	1	15	3
13	2	15	2	16	2
14	2	17	1	17	3
15	1	18	2	18	1
17	2	21	1	19	1
18	3	25	1	22	3
19	4	27	1	23	1
20	1	29	2	24	1
21	4	30	2	25	2
23	2	31	3	26	1
24	1	32	1	28	1
25	1	33	3	29	1
26	2	34	1	30	2
27	1	35	3	31	1
28	2	38	3	32	1
31	1	40	1	35	3
34	1	43	1	38	1
43	1	44	1	39	1
		103	1	42	1
				43	1
				49	3
				50	1
				51	1
				57	1
				66	1
				70	1
				81	1
				164	1

Table 3.2 Detected faults in GNOME official public releases

GNOME 2.0		GNOME 2.2		GNOME 2.4	
Weeks from release	Detected bugs	Weeks from release	Detected bugs	Weeks from release	Detected bugs
1	6	1	5	1	4
2	5	2	4	2	5
3	3	3	5	3	2
4	2	4	5	4	7
5	5	5	9	5	3
6	5	6	5	6	1
7	8	7	2	7	3
8	4	8	1	8	4
9	8	9	2	9	3
10	3	10	3	10	5
11	2	11	2	11	1
12	1	13	1	12	3
13	6	15	4	15	2
14	8	16	1	18	1
15	6	17	1	19	1
16	2	18	1	20	5
17	2	22	1	21	2
18	1	24	2	23	1
19	1			46	1
20	1				
21	1				
22	2				
24	3				

3.4.2 Reliability Assessment for Open Source Software

To compare the proposed model with traditional models for reliability assessment, the widely used GO model (Goel and Okumoto, 1979) and S-shaped model (Yamada et al., 1983) are selected as examples of traditional models. The mean value functions of these two models for release *i* are

$$m_i(t) = a_i\left[1 - \exp\left(-b_i t\right)\right] \text{ and } m_i(t) = a_i\left[1 - \left(1 + b_i t\right)\exp\left(-b_i t\right)\right], \tag{3.11}$$

where a_i represents the number of expected initial faults in each release i. Furthermore, their corresponding fault detection rate functions are

$$b_i(t) = b_i \text{ and } b_i(t) = b_i^2 t / (1 + b_i t). \tag{3.12}$$

Since both GO model and S-shaped model are based on the assumption that $a_i(t)=a_i$, this assumption is also adopted in our proposed model. Hence, the mean value function of the proposed model for release i is obtained as

$$m_i(t) = a_i \left\{1 - \exp\left[-B_i^*(t)\right]\right\} \tag{3.13}$$

where $B_i^*(t) = B_i(t) - B_i(0)$ and $B_i(t)$ is the integration of $b_i(t)$ over the time period $(0, t]$.

Parameters of these models are estimated by the least square estimation (LSE) method. The estimation is done by minimizing the sum of squared residuals, which is the difference between estimated values and true observations as $\sum_{i=1}^{k} \sum_{j=1}^{k_i} \left[m_i(t_{ij}) - n_{ij}\right]^2$, where n_{ij} denotes the cumulative number of detected faults until time t_{ij}, i denotes the release number and j denotes the observation number for each release i. Specifically, i=1, 2, …, k and j=1, 2, …, k_i. It means that there are totally k releases, and for each release i, there are k_i number of observations.

With the LSE method, the descriptive performance of the model can be measured by the mean squared error (*MSE_i*) for each release *i*, and if it is small, it indicates the good descriptive performance of the model. In particular, the MSE_i is given by

$$MSE_i = \frac{1}{k_i}\left\{\sum_{j=1}^{k_i}\left[m_i\left(t_{ij}\right) - n_{ij}\right]^2\right\}. \tag{3.14}$$

After the estimates of the parameters are obtained numerically, these models can be used to measure the reliability of the software. Generally, software reliability at current time *t* is measured by

$$R(x \mid t) = \exp\left[-\left\{m(t+x) - m(t)\right\}\right]. \tag{3.15}$$

In (3.15), $R(x \mid t)$ represents the conditional software reliability which is defined as the probability that the software will not fail given a specified time interval $(t, t+x]$ (Musa et al. 1987; Xie, 1991). However, $R(x \mid t)$ here cannot measure the reliability for OSS accurately. The reason is coming from the unique property of OSS: the hump-shaped fault detection rate function. More specifically, suppose that most volunteers have left from a specific release of OSS, no matter how many remaining faults are still in this release, this release can generate a high value of $R(x \mid t)$. In this case, the software may not be really reliable as the high reliability is due to the fact that few people are using it. With the consideration of this, we adopt the reliability measurement as shown in (3.8). Although it is simple, it can assess the OSS reliability more precisely.

In order to compare the reliability on the same time basis for different models, the real version-update time for each release is used. For Apache data, the real release times are T_{r1}=32, T_{r2}=41, and T_{r3}=53 days from each release respectively; while for GNOME data, these numbers are 32, 31 and 29 weeks. Estimated parameter values and numerical results are shown in Table 3.3 and Table 3.4.

Table 3.3 Estimated parameter values and numerical results for Apache

No. of release	Different models	Estimated parameters	MSE_i	$R_i(T_{ri})$
1	GO model	a_1=84.60, b_1=0.0564	5.76	0.8352
1	S-shaped model	a_1=74.27, b_1=0.1539	7.70	0.9570
1	Proposed model	a_1=106.04, N_1=1.6798, A_1=3.1910, $\alpha_1 = 0.1055$	2.80	0.6717
2	GO model	a_2=52.32, b_2=0.0393	8.84	0.8007
2	S-shaped model	a_2=49.90, b_2=0.0896	8.39	0.8813
2	Proposed model	a_2=88.19, N_2=1.0746, A_2=3.5814, $\alpha_2 = 0.0740$	5.98	0.4944
3	GO model	a_3=58.38, b_3=0.0367	2.57	0.8571
3	S-shaped model	a_3=56.90, b_3=0.0805	2.40	0.9260
3	Proposed model	a_3=82.97, N_3=1.5645, A_3=3.0012, $\alpha_3 = 0.0576$	0.68	0.6245

Table 3.4 Estimated parameter values and numerical results for GNOME

No. of release	Different models	Estimated parameters	MSE_i	$R_i(T_{ri})$
1	GO model	a_1=140.09, b_1=0.0418	11.84	0.7371
1	S-shaped model	a_1=90.58, b_1=0.1818	7.47	0.9797
1	Proposed model	a_1=142.06, N_1=1.1538, A_1=5.9508, $\alpha_1 = 0.1794$	4.44	0.6195
2	GO model	a_2=55.98, b_2=0.1255	2.93	0.9796
2	S-shaped model	a_2=50.78, b_2=0.3276	4.12	0.9996
2	Proposed model	a_2=70.76, N_2=2.1194, A_2=1.9606, $\alpha_2 = 0.1735$	2.44	0.7496
3	GO model	a_3=55.17, b_3=0.1003	2.92	0.9455
3	S-shaped model	a_3=52.86, b_3=0.2302	4.32	0.9903
3	Proposed model	a_3=68.99, N_3=2.1343, A_3=2.4748, $\alpha_3 = 0.1378$	1.99	0.7600

It can be seen that for different releases of Apache and GNOME, the proposed model has the best descriptive performance with the smallest value of MSE_i. The proposed model can describe the failure process of OSS more accurately. Furthermore, in the later stage of software testing, there are fewer and fewer faults detected. Since both GO model and S-shaped model cannot describe the hump-shaped fault detection rate function accurately, they describe this with the assumption that most faults in the software have already been detected. Therefore, they provide an underestimation of the number of expected initial faults in the software and an overestimation of the reliability of software. The estimates of the reliability of software from traditional models are especially dangerous for management as they could be too optimistic to be acceptable.

3.4.3 A Decision Model Application Example

For management, it is of equal importance to predict the optimal version-update time for each release. It should be noted here that the version-update time is a more accurate concept than the release time for OSS. The reason lies in the fact that software is still used and tested by the volunteers after the each version-update. In other words, even after the version-update of OSS, it is still under the testing phase unless there is other information to indicate that this OSS is released for commercial use. Due to this consideration, the failure data after each version-update is also used as shown in Table 3.1 and Table 3.2. Specifically, in this part, the decision model is validated on the first release of Apache. Based on the procedure discussed in Section 3.3, the determination of the optimal version-update time is presented as in the following steps.

Step 1: Quantification of rapid release indicator and reliability

As discussed, rapid release indicator β_1 and reliability R_1 are two important factors for management to determine the optimal version-update time for the first release of Apache. Based on the failure data shown in Table 3.1, the model parameters can be estimated as shown in Table 3.3. Then, both of these two attributes are quantitatively measured by (3.7) and (3.8) and our decision space is $[t_{1\max}, +\infty)$ where $t_{1\max} = 11$.

Step2: Elicitation of single utility function for each attribute

The single utility function for each attribute is elicited based on the management's own scenarios. Since these management scenarios are subjective assessments from management, they may not be precise. In this case, sensitivity analysis is needed, and it will be discussed in the next subsection.

Suppose that management scenarios in our application example are as follows:

(1) Management demonstrates its risk neutral attitude for each attribute.

(2) Under the rapid release strategy, management indicates that at least half of the maximum testing effort from volunteers at $t_{1\max}$ should be maintained and the larger the better; the highest rapid release expectation is achieved at the time when the maximum testing effort from volunteers is reached.

(3) Considering the reliability of software, management has verified that at least 10% of software faults should be detected and the more the better; its highest reliability expectation is achieved when 60% of software faults are detected.

According to the scenarios above, some important points on the utility curve are obtained. In particular, the lowest rapid release requirement is $\beta_1^0 = 0.5$ and the highest rapid release expectation is $\beta_1^1 = 1$; the lowest reliability requirement is $R_1^0 = 0.1$ and the highest reliability expectation $R_1^1 = 0.6$. Additionally, based on management's risk neutral attitude towards these two attributes, the linear form of the single utility function should be used. Specifically, we have $u(\beta_1) = 2\beta_1 - 1$ and $u(R_1) = 2R_1 - 0.2$. It is worth noting here that although the linear form is simple, it is a widely accepted form especially when empirical results are needed (Scholz and Tietje, 2002).

Step 3: Estimation of scaling constants

In this stage, the scaling constant w_{β_1} is estimated first by comparing the two choices in Figure 3.1. Management has claimed that it is indifferent between these two choices when *p* is equal to 0.5. Therefore, $w_{\beta_1} = 0.5$. Since the sum of scaling constants is equal to one, w_{R_1} is equal to 0.5 as well.

Step 4: Maximization of multi-attribute utility function

Finally, based on the estimated single utility functions and the scaling constants, the multi-attribute utility function is evaluated and it is shown in Figure 3.3. The multi-attribute utility function is maximized at the optimal version-update time $T_1^* = 15.32$. It means that Apache release one should be updated at this time, under which the conjoint outcome of $\beta_1\left(T_1^*\right) = 0.95$ and $R_1\left(T_1^*\right) = 0.47$ can provide the greatest overall satisfaction for management. Given that the real version-update time is T_{r1}=32, a delayed version-updating is used in practice under the provided management scenarios. More specifically, when the real version-update time is used, we have $\beta_1\left(T_{r1}\right) = 0.35$ and $R_1\left(T_{r1}\right) = 0.67$. Although 67% of total software faults in the release are detected, 65% of the maximum testing effort from volunteers is lost.

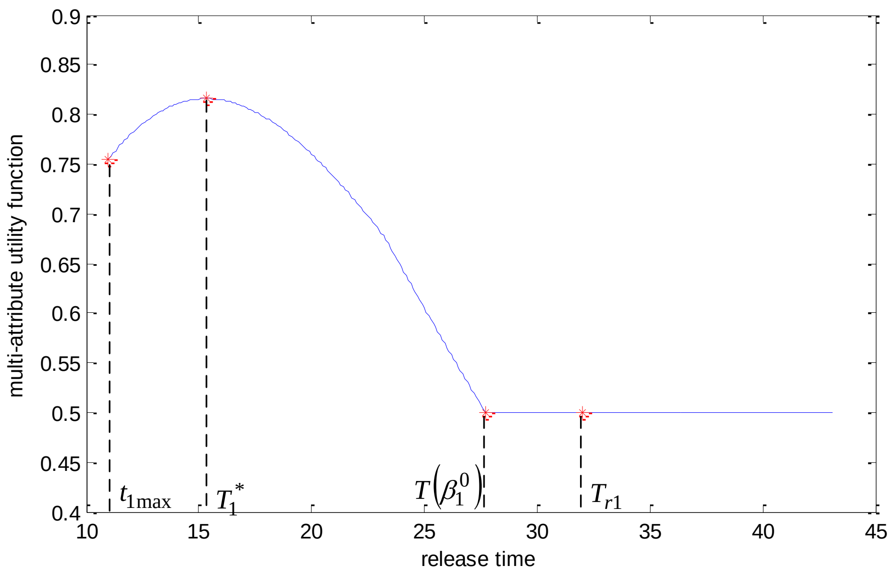


Figure 3.3 The multi-attribute utility function given different release times

It is worth noting that in Figure 3.3, $t_{1\max} = 11$ denotes that $b_1(t)$ reaches its maximum at this time. In other words, testing effort from volunteers reaches the maximum at this time and will decrease from this time on. If the OSS is updated at this time, the highest rapid release expectation $\beta_1^1 = 1$ is satisfied. However, at this time, the reliability is low and we have $R_1\left(t_{1\max}\right) = 0.36$. It means that only 36% of total faults in the software are detected. Due to this consideration, software is expected to be tested longer for a higher reliability.

In addition, we denote $T\left(\beta_1^0\right)$ as the time when the lowest rapid release requirement $\beta_1^0 = 0.5$ is reached and we have $T\left(\beta_1^0\right) = 27.7$. Figure 3.3 shows that the multi-attribute utility function remains at the 0.5 level when the version-update time is greater than $T\left(\beta_1^0\right)$. It indicates that only the reliability of software has reached its highest expectation level $R_1^1 = 0.6$ from this time on. However, at the same time period, software performs not well in terms of the rapid release requirement.

Based on the discussions above, it can be seen that when both reliability and rapid release strategy are considered, a compromise should be between $t_{1\max} = 11$ and $T\left(\beta_1^0\right)$. Figure 3.3 has shown us that the overall satisfaction level is maximized at the optimal version-update time $T_1^* = 15.32$.

3.4.4 Sensitivity Analysis

Optimal version-update time for each release *i* can be determined by maximizing the multi-attribute function. However, since most parameters in the MAUT are obtained based on the subjective assessments from management, the optimal version-update time received may not be precise; in practice, sophisticated management needs to know how robust the result is. Accordingly, sensitivity analysis is needed.

Sensitivity analysis is generally done by changing one parameter and setting the other parameters at their fixed values (Xie and Hong, 1998, Huang and Lyu, 2005b; Lo et al., 2005; Huang and Lo, 2006; Yang et al., 2008; Li et al., 2010). When the parameter θ is investigated to see how much the optimal version-update time T_i^* is changed, T_i^* is in fact a function of θ as other parameters are fixed using their estimated values. Then $S_{q,\theta}^i$ can be calculated and it is defined as the relative change of the optimal version-update time for release *i* when θ is changed by 100*q*%. That is

$$S_{q,\theta}^i = \left| \frac{T_i^*(\theta + q\theta) - T_i^*(\theta)}{T_i^*(\theta)} \right| \tag{3.16}$$

A large value of $S_{q,\theta}^i$ indicates that parameter θ has a significant influence on the determination of T_i^*. Equivalently speaking, T_i^* is regarded as sensitive to the change of θ. Normally, management should pay special attention to significant parameters (Xie and Hong, 1998, Huang and Lyu, 2005b; Lo et al., 2005; Huang and Lo, 2006; Yang et al., 2008; Li et al., 2010).

Based on the decision model application example, results of sensitivity analysis are shown in Table 3.5. It is worth noting that the highest rapid release expectation is achieved when the maximum testing effort from volunteers is reached. Mathematically, it means that software should be released at $t_{i\max}$ when $b_{i\max}$ is achieved and $\beta_1^1 = 1$. Therefore, based on (3.7), it can be seen that the positive change of the highest rapid release indicator β_1^1 is impossible. In addition, w_{R_1} is not investigated in the sensitivity analysis because the sum of scaling constants is always equal to one.

Table 3.5 Results from sensitivity analysis

q	-30%	-20%	-10%	10%	20%	30%
$S^i_{q,w_{\beta_1}}$	0.197	0.118	0.054	0.046	0.085	0.120
S^i_{q,β_1^0}	0.072	0.048	0.024	0.025	0.050	0.076
S^i_{q,β_1^1}	0.479	0.313	0.123	NA	NA	NA
S^i_{q,R_1^0}	0.014	0.010	0.005	0.005	0.010	0.016
S^i_{q,R_1^1}	0.126	0.041	0.033	0.027	0.049	0.067

Table 3.5 indicates that parameters β_1^0 and R_1^0 are not significant parameters. For example, when β_1^0 changes by 30%, the relative change of T_1^* is still less than 8%. From management's point of view, it means that its lowest rapid release and reliability requirements will not have a significant effect on the final decision of optimal version-updating. More specifically, requirements, such as (a) at least half of the maximum testing effort from volunteers should be maintained and (b) at least 10% of faults should be detected, are not significant. Accordingly, it is not necessary for management to reassess these requirements.

On the other hand, parameters w_{β_1} and β_1^1 are significant parameters and management needs to pay special attention to them. Normally, reassessments about these parameters are needed for more accurate estimates. In particular, for the importance weight w_{β_1} allocated for the rapid release indicator, management should check the probability p in Figure 3.1 again. As to the highest rapid release expectation β_1^1, management should reassess whether it is achieved when the maximum testing effort from volunteers is reached.

One special parameter is R_1^1 which represents the highest reliability expectation from management. Sensitivity analysis results indicate that the positive change of it does not affect the final decision much; while more than 20% negative change of it could significantly affect the decision on optimal version-updating. Therefore, management should be asked about whether its highest satisfaction towards reliability can be only achieved when more than 60% of faults are removed from the software. If this is the case, no more reassessments about the highest reliability expectation are needed; otherwise, the highest reliability expectation should be checked again, especially for a large decrease of this expectation.

3.5 Conclusion

The OSS approach provides a new paradigm of software development, where volunteer participation has become a critical issue. Since volunteers are interest-driven and the attractiveness of a specific release of software is generally decreasing over time, multiple releases are expected to maintain a sufficient number of volunteers and to attract new comers. In order to describe these unique properties of OSS properly, a modified NHPP model is proposed to assess OSS reliability. Based on the numerical results, it is found that traditional models provide too optimistic reliability estimates.

Furthermore, since multiple releases of OSS are common, and it is often imperative to know when to conduct the version-updating in the testing phase. On one hand, with the consideration of volunteers' participation, software is expected to release as early as possible when the volunteers' testing effort involved in OSS reaches its maximum. On the other hand, reliability is also important because it is the most important aspect of software quality. With the consideration of OSS reliability, software should be tested for a long time prior to the next version-updating. The difficulty is that rapid release strategy and OSS reliability are contradicting with each other. In order to make a judicious decision on the optimal version-updating in this case, a decision model based on MAUT is proposed. The application example has shown that the proposed decision model can assist management to make a rational decision based on its own scenarios. Our future research will investigate more OSS projects to justify the generality of our proposed model for OSS reliability and its optimal version-updating.

However, there are some weaknesses to our proposed decision model for the determination of optimal version updating for OSS. Although sensitivity analysis can help management to determine what significant parameters are and more attention can be paid to them, the overall decision process is still quite subjective. Therefore, experts' past experience and historical data are important for management to obtain a trustworthy estimated optimal version update time. This is an interesting research direction that can be explored in the future.

In addition, we only consider reliability and rapid release strategy in our approach. In reality, there could be other attributes that should be incorporated in the decision model. For example, when we desire for a rapid release of OSS, it will inevitably increase the number of software versions, and a corresponding increase in the complexity of the software product. More specifically, Eclipse was plagued with compatibility problems due to the rapid release of software. In this case, complexity can be added as another attribute that should be considered in the decision model. Future research on this kind of problem will further refine our proposed decision model.

Chapter 4 Performance Improvement for DDSRMs

4.1 Basic Problem Description

Most of recent DDSRMs are based on multiple-delayed-input single-output (MDISO) architecture (Cai et al., 2001; Tian and Noore, 2005a; Tian and Noore 2005b; Pai and Hong, 2006; Hu et al., 2007; Yang and Li, 2007; Yang et al., 2007). DDSRMs with MDISO architecture form an important class of existing DDSRMs, which are focused on the inter-relationship among software failure data instead of the relationship between failure sequence number and failure data. In this chapter, our research also focuses on DDSRMs with MDISO architecture, and we refer to the term “DDSRM” as “DDSRM with MDISO architecture” if no further explanation is given.

In existing data-driven approach to software reliability modeling and prediction, the software failure process is viewed as a time series. The inputs used by a DDSRM are the past, consecutive lagged observations of the time series, while the outputs are the future value. The time series model used by existing DDSRMs can be represented as follows.

$$x(i) = F[x(i-1), x(i-2), \cdots, x(i-w)], \tag{4.1}$$

where $x(\cdot)$ is the observation from the software failure process, e.g., cumulative numbers of detected software faults (Hu et al., 2007), software failure times (Tian and

Noore; 2005a; Tian and Noore, 2005b), inter-failure times (Cai 2001; Pai and Hong, 2006), etc.; $[x(i-1), x(i-2), \cdots, x(i-w)]$ is a vector of consecutive lagged terms taken from the time series; and $F(\cdot)$ is the time series model describing the relationship between the past observations and the future value. In the literature, $x(i)$ is also denoted by x_i for simplicity. In (4.1), *w* is the dimension of the input vector, which is also termed as the size of the sliding window (Hu et al., 2007), the fixed-length of moving window (Chen, 2007), the order of autoregressive terms (Chen, 2007), etc.

The processes of using a DDSRM for software reliability modeling and prediction are illustrated in Figure 4.1, which consist of a *training process*, a *testing process*, and a *prediction process*. Suppose that we have observed a total number of *n* software failures, and failure data $\{x_i, i = 1,2,\cdots,n\}$ are recorded; then a DDSRM can be constructed. The constructed model will first be trained with the first $(n-d)$ failure data, $\{x_i, i = 1,2,\cdots,n-d\}$, where *d* is a nonnegative integer determined by the model user; then the trained model will be tested for model performance by the rest *d* failure data, $\{x_i, i = n-d+1, n-d+2, \cdots, n\}$. If the testing result of model performance is satisfactory, then the trained model can be used for prediction purpose.

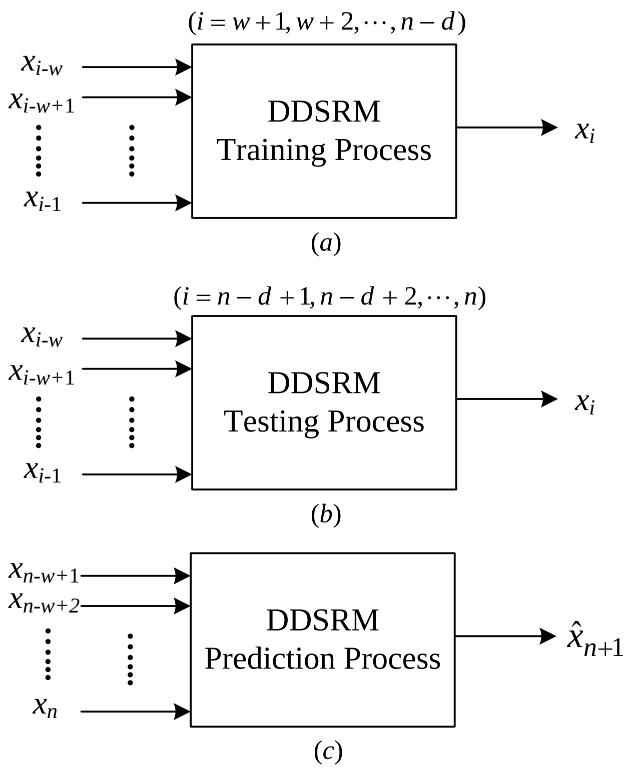


Figure 4.1 The processes of using a DDSRM for software reliability modeling and prediction

During the model training process, a w-dimensional vector, $\{x_{i-w}, x_{i-w+1}, \cdots, x_{i-1}\}$, is used as model input and x_i is used as the expected output, which form a training sample pattern, as illustrated in Figure 4.1 (a). With i changing from $(w+1)$ to $(n-d)$, there are a total of $(n-d-w)$ training sample patterns which are fed into the DDSRM (Chen, 2007), as shown in Table 4.1. The objective of the model training process is to make the model have the best fitting of recorded data, i.e., having the smallest *mean squared error of data-fitting* (MSE_f), defined as

$$MSE_f \equiv \frac{1}{n-d-w} \sum_{i=w+1}^{n-d} (\hat{x}_i - x_i)^2, \tag{4.2}$$

where $\hat{x}_i$ is the output obtained from the DDSRM (estimated observation), and x_i is the recorded failure datum (true observation).

Table 4.1 Sample patterns used in model training process

	Model Input	Expected Output
$i = w+1$	$\{x_1, x_2, \cdots, x_w\}$	x_{w+1}
$i = w+2$	$\{x_2, x_3, \cdots, x_{w+1}\}$	x_{w+2}
$\vdots$	$\vdots$	$\vdots$
$i = n-d$	$\{x_{n-d-w}, x_{n-d-w+1}, \cdots, x_{n-d-1}\}$	x_{n-d}

After the training process, the model has "learnt" the inherent patterns of the software failure process; however, as the model will be used for prediction purpose, its prediction accuracy needs to be tested before it can be practically used. During the model testing process (it is also called validation process), a *w*-dimensional vector $\{x_{i-w}, x_{i-w+1}, \cdots, x_{i-1}\}$ is fed into the trained model as input, and the output, $\hat{x}_i$, the predicted value by the trained model, is obtained, as illustrated in Figure 4.1 (*b*). With i changing from $(n-d+1)$ to *n*, there are a total of *d* predicted values obtained. Model performance can be measured by the *mean squared error of prediction* (*MSE_p*), defined as

$$MSE_p = \frac{1}{d} \sum_{i=n-d+1}^{n} (\hat{x}_i - x_i)^2 . \tag{4.3}$$

If the obtained MSE_p is at an acceptable level, which implies that the trained model has satisfactory performance, then the model can be used for prediction of future failure. During the prediction process, the most recent *w* failure data, $\{x_{n-w+1}, x_{n-w+2}, \cdots, x_n\}$, are used as model input, and the model can then give the predicted value of $\hat{x}_{n+1}$, as illustrated in Figure 4.1 (*c*). The above model training, testing, and prediction processes are carried out iteratively at the observation of each new software failure. For example, when the (*n*+1):th software failure is observed, and thus failure datum x_{n+1} becomes available, the model will be once again trained and tested using new failure data set, $\{x_i, i = 1, 2, \cdots, n+1\}$, and then it can be used to give the prediction of $\hat{x}_{n+2}$.

From above discussions, it can be seen that existing DDSRMs assume that a software failure is strongly correlated with the most recent *w* failures, see equation (4.1); however, this assumption may not be valid in reality because in a time series the correlation can be quite complicated (Tsay, 2002). In fact, it is possible that a software failure is correlated with *some* of previous failures, not necessarily being the *most recent* ones. For example, a failure, x_i, could be correlated with, say, x_{i-8}, x_{i-6}, and x_{i-2}. If this is the case, these three time lag terms should be used as model inputs. This issue, despite its importance, has not been addressed in the literature.

In this chapter, we relax the unrealistic assumption adopted by existing DDSRMs and develop a generic DDSRM. Existing DDSRMs are special cases of the proposed model. We also develop a GA-based algorithm to discover the correlation among software failures, by which appropriate time lag terms can be determined to be used as the inputs of the proposed DDSRM. Numerical examples are presented to testify the validity of the proposed model and algorithm.

The remainder of this chapter is organized as follows. In Section 4.2, since we take SVM-based DDSRMs as the illustrating example, the basic theory of SVM for regression is briefly reviewed. In Section 4.3, a new DDSRM is developed and a GA-based algorithm is proposed. Numerical examples are presented in Section 4.4. In Section 4.5, some concluding remarks are made.

4.2 A Brief Review of SVM for Regression

As our numerical examples will use SVM-based SRMs, a major class of DDSRMs, here we give a brief review of SVM for regression. Interested readers can refer to Vapnik (1995), Vapnik (1999), Kecman (2001), Scholkopf and Smola (2002) for more detailed discussions.

In general, SVMs can be used for two purposes, i.e., classification and regression. SVM-based SRMs are constructed by SVM for regression. Vapnik (1995) introduced a regression function which can reflect the mapping of input and output of a process by learning a set of training data, $\{(x_i, y_i)\}_{i=1}^{l}$, where x_i's are the actual values of the input vectors and y_i's are the actual values of the output, l is the number of total data pairs. Based on the structural risk minimization principle, the SVM regression minimizes an upper bound on the expected risk.

Unlike traditional empirical risk minimization which attempts to minimize the error on the training data, minimizing this bound could achieve high generalization performance (Vapnik, 1995; Vapnik, 1999). The SVM model used for regression function is given by

$$f(x) = \omega\phi(x) + b, \tag{4.4}$$

where $\phi(x)$ denotes the feature space which is transformation of the input space x. In other words, ϕ is the high-dimensional feature space mapping function. By equation (4.4), the nonlinear relationship of the input and the output in the low-dimensional space can be written in a linear form in the high-dimensional feature space (Vapnik, 1995; Vapnik, 1999); and the "dimension disaster" problem can be overcome following the above specific transformation in SVM regression. The coefficients ω and b can be determined by minimizing the following regularized risk function

$$R = \frac{1}{2}\|\omega\|^2 + C\frac{1}{l}\sum_{i=1}^{l}|y_i - f(x_i)|_\varepsilon, \tag{4.5}$$

where

$$|y_i - f(x_i)|_\varepsilon = \begin{cases} 0 & if\ |y_i - f(x_i)| \le \varepsilon \\ |y_i - f(x_i)| & \text{otherwise} \end{cases}. \tag{4.6}$$

In the above regularized risk function, C is the regulation constant which represents the trade-off between model structure complexity $\frac{1}{2}\|\omega\|^2$ and empirical error $\frac{1}{l}\sum_{i=1}^{l}|y_i - f(x_i)|_\varepsilon$. By minimizing this risk function, structural risk minimization can be achieved, which in turn improves the model generalization capability. Furthermore, to define the ε-insensitive linear loss function more clearly, as shown in Figure 4.2, two slack variables, ξ and ξ^*, which represent the difference between the estimated value and the real value, are given by

$$|y_i - f(x_i)| - \varepsilon = \begin{cases} \xi & if \;\; y_i - f(x_i) > \varepsilon \\ \xi^* & if \;\; f(x_i) - y_i > \varepsilon \end{cases} . \tag{4.7}$$

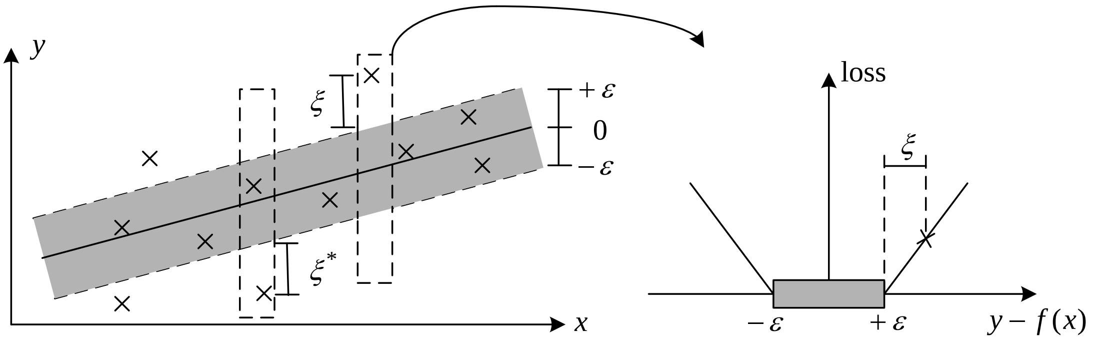


Figure 4.2 The soft margin loss setting for a linear SVM regression (Scholkopf and Smola, 2002)

In this case, the regularized risk function can be written in another form, which is

$$R = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi + \xi^*), \tag{4.8}$$

where

$$\begin{cases} y_i - \omega\phi(x_i) - b \le \varepsilon + \xi_i & i = 1,2,\cdots,l \\ \omega\phi(x_i) + b - y_i \le \varepsilon + \xi_i^* & i = 1,2,\cdots,l\,. \\ \xi, \xi^* \ge 0 & i = 1,2,\cdots,l \end{cases} \tag{4.9}$$

In general, minimizing the above regularized risk function directly is cumbersome and inefficient. Alternatively, according to Karush-Kuhn-Tucker conditions, this optimization problem can be transformed into maximizing its dual Lagrangian problem, which is given by

$$\begin{aligned} L(\alpha_i, \alpha_i^*) = {} & \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) - \varepsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) \\ & - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(x_i, x_j) \end{aligned}, \tag{4.10}$$

subject to constraints

$$\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0\,,$$

$$0 \le \alpha_i, \alpha_i^* \le C \quad i = 1,2,\cdots,l\,.$$

In the dual Lagrangian form, $K(x_i, x_j)$ is the *kernel function*. In practice, polynomial and Gaussian kernel functions are commonly used. α_i's and α_i^*'s are Lagrange multipliers which satisfy $\alpha_i \cdot \alpha_i^* = 0$ for $i = 1,\cdots,l$. After α_i's and α_i^*'s are obtained, the regression function can be rewritten as

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) K(x_i, x_j) + b\,. \tag{4.11}$$

Similar to Tian and Noore (2005b), Pai (2006), Pai and Hong (2006), Chen (2007), Yang and Li (2007), Yang et al. (2007), we adopt Gaussian kernel function in our research, which is given by

$$K(x_i, x_j) = \exp\left[\frac{-(x_i - x_j)^2}{2\sigma^2}\right]. \tag{4.12}$$

Substitute (4.12) into (4.10), and by maximizing $L(\alpha_i, \alpha_i^*)$ with constraints, α_i's and α_i^*'s can be obtained. Substitute them into (4.11), and the SVM regression function $f(x)$ is obtained.

4.3 A Generic DDSRM with a Hybrid GA-Based Algorithm

As discussed, existing DDSRMs seem to have a fundamental drawback. For these models, it is assumed that a software failure is strongly correlated with the most recent *w* failures; however, this assumption may not be valid in reality. In a time series, the correlation may be quite complicated, thus it is more reasonable to assume that a failure is strongly correlated with *some* of previous failures as follows.

$$x(i) = F[x(i - m_1), x(i - m_2), \cdots, x(i - m_p)], \tag{4.13}$$

where $x(\cdot)$ is the quantity of interest; $x(i-m_1), x(i-m_2), \cdots, x(i-m_p)$ are time lag terms taken from the time series; and $F(\cdot)$ is the time series model. It can be seen that (4.13) is a more general time series model, and the normally adopted model, (4.1), is a special case of (4.13) for which $p = w$ and $m_i = i(1 \le i \le p)$.

Before the time series model given by (4.13) can be used, the model user first needs to determine the time lag terms that should be used, i.e., to determine the value of *p* and the values of $m_1, m_2, \cdots, m_p$. This is actually to discover the inherent correlation of observations (i.e., software failures) in the time series. For linear time series analysis, this can be done by using autocorrelation function (ACF), which can be calculated from the observed data (Tsay, 2002). If a time series has significant ACF at, say, lags 2, 6, and 8, then the time series model should be $x_i = F(x_{i-2}, x_{i-6}, x_{i-8})$ in this particular case. However, for DDSRMs based on ANNs and SVMs, which are non-linear time series models, the method to determine the time lag terms to be used as model inputs is yet to be developed.

In time series analysis, whenever the time series model takes a different set of time lag terms as the input, it could be viewed as a new model. Therefore, to determine the appropriate time lag terms to be used could be thought of as to determine the appropriate time series model. For this reason, this kind of exploring process is sometimes referred to as a *model mining* process. In the literature, GA-based algorithms have been developed to conduct model mining, which have proved to be effective and efficient (Valdes and Mateescu, 2002; Valdes and Bonham-Carter, 2006).

In the model mining process, for the sake of easy representation and programming, the time lag terms that should be used are represented by a binary code (Valdes and Mateescu, 2002; Valdes and Bonham-Carter, 2006), the length of which is denoted by *v*, which is a nonnegative integer determined by the model user. The position of a value of "1" in the binary code indicates a time lag term that should be used. For example, the binary code "10100010" (for which $v = 8$ and $p = 3$) means that x_{i-2}, x_{i-6}, and x_{i-8} should be used, as shown in Figure 4.3.

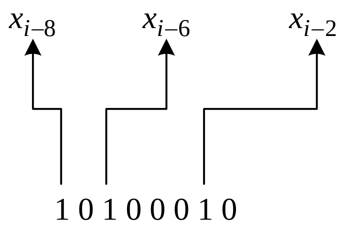
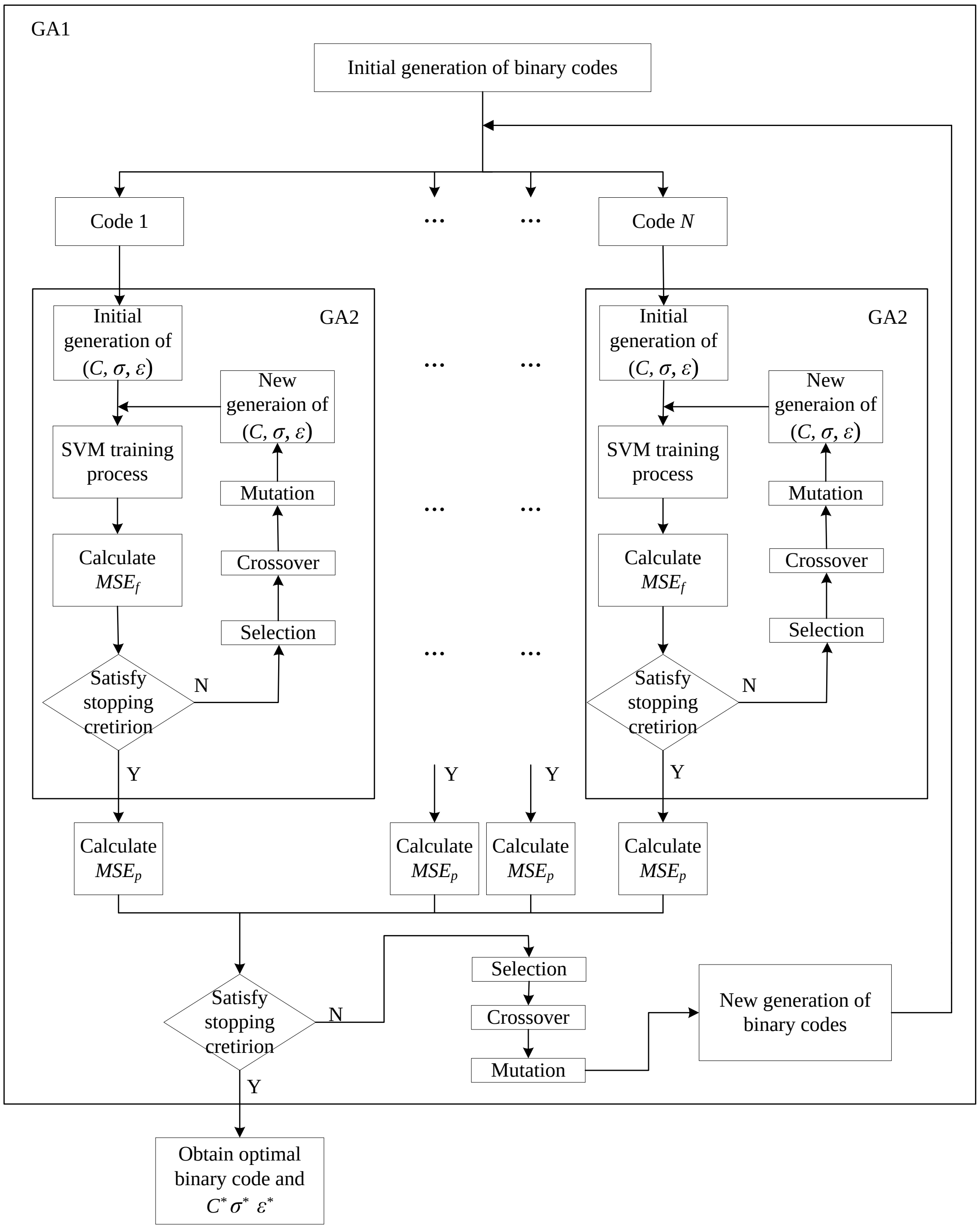


Figure 4.3 Interpretation of a binary code in model mining

Besides the determination of the time lag terms that should be used as model inputs, for a DDSRM, the determination of model parameter values is of equal importance as it has great impact on model performance (Pai, 2006; Pai and Hong, 2006; Chen, 2007). Taking both issues into consideration, we develop a hybrid GA-based algorithm by which the time lag terms to be used as well as the optimal values of model parameters can be determined simultaneously.

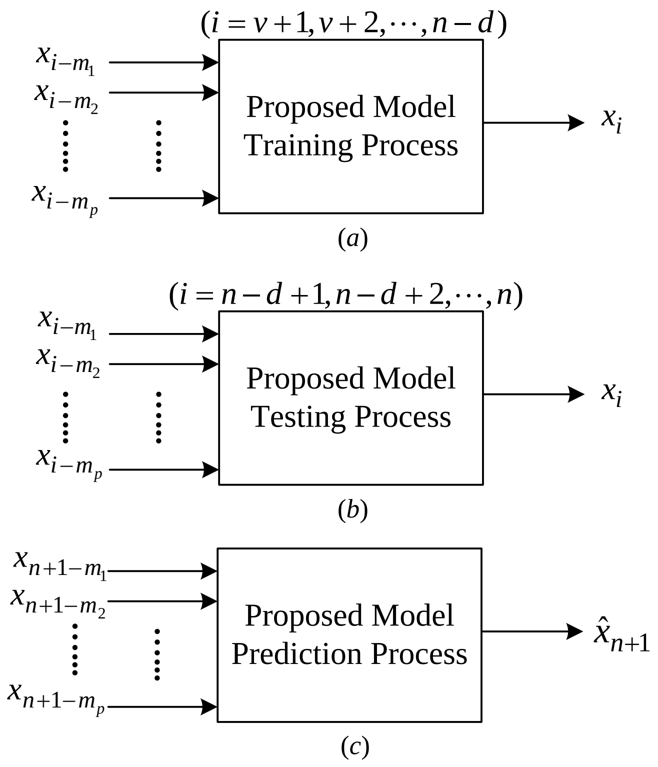
The developed algorithm is shown in Figure 4.4, detailed explanation of which will be given later. In Figure 4.4, we take SVM-based SRM with Gaussian kernel function as an example of DDSRMs; however, the idea behind is applicable to other DDSRMs such as those based on ANNs, provided that GA2 is modified accordingly.



GA1
Initial generation of binary codes
Code 1
…
…
Code N
GA2
Initial generation of (C, σ, ε)
New generaion of (C, σ, ε)
SVM training process
Mutation
Calculate MSE_f
Crossover
Selection
Satisfy stopping cretirion
N
Y
GA2
Initial generation of (C, σ, ε)
New generaion of (C, σ, ε)
SVM training process
Mutation
Calculate MSE_f
Crossover
Selection
Satisfy stopping cretirion
N
Y
Y
Y
Calculate MSE_p
Calculate MSE_p
Calculate MSE_p
Calculate MSE_p
Satisfy stopping cretirion
N
Y
Selection
Crossover
Mutation
New generation of binary codes
Obtain optimal binary code and $C^* \sigma^* \varepsilon^*$

Figure 4.4 A hybrid GA-based algorithm to determine the time lag terms to be used and the optimal parameters of SVM-based SRM with Gaussian kernel function

Based on above discussions, we propose a new DDSRM which takes time series model (4.13) instead of (4.1), and uses the developed hybrid GA-based algorithm in Figure 4.4 to find the time lag terms that should be used as well as the optimal values of model parameters. The proposed DDSRM is illustrated in Figure 4.5. Note that there exists fundamental difference between the proposed DDSRM and existing DDSRMs. In Figure 4.1, the most recent *w* failure data are used as model inputs; while in Figure 4.5, time lag terms identified by the developed hybrid GA-based algorithm during the model training and testing processes are used.



$(i = v+1, v+2, \cdots, n-d)$
x_{i-m_1}
x_{i-m_2}
x_{i-m_p}
Proposed Model Training Process
x_i
(a)
$(i = n-d+1, n-d+2, \cdots, n)$
x_{i-m_1}
x_{i-m_2}
x_{i-m_p}
Proposed Model Testing Process
x_i
(b)
x_{n+1-m_1}
x_{n+1-m_2}
x_{n+1-m_p}
Proposed Model Prediction Process
$\hat{x}_{n+1}$
(c)

Figure 4.5 The processes of using the proposed DDSRM

The developed hybrid GA-based algorithm in Figure 4.4 consists of five steps, which are described below.

Step 1: Generation of chromosomes

The time lag terms used in (4.13) are expressed by a binary code e.g., the binary code "10100010" indicates that x_{i-2}, x_{i-6}, and x_{i-8} are used as model input (see Figure 4.3). The first GA (GA1 in Figure 4.4) generates an initial generation of *N* randomly selected binary codes. Generally, the value of *N* is much less than 2^v which is the number of all possible combinations of binary codes. In the framework of GA, these binary codes are termed as chromosomes.

Step 2: Local optimization of three parameters in SVM

Under each chromosome, a second GA (GA2 in Figure 4.4) is introduced to determine the optimal values of three parameters in SVM. Following the general way of applying GA, an initial generation of chromosomes is generated, each chromosome is a set of parameter values, i.e., (C, σ, ε). Then, the failure data are fed into the SVM training process. After the training process, each chromosome is tested by its fitness function value, which is the MSE_f defined by (4.2) (note that *w* in (4.2) should be replaced by *v*). If the stopping criterion of GA2, which could be that the MSE_f is minimized or a predetermined number of generations is reached, is satisfied, then GA2 is completed and the algorithm returns to GA1; otherwise the chromosomes that have small fitness function values are selected and the crossover and mutation are conducted, thus an offspring generation is generated, and the algorithm goes back to the SVM training process under this new generation of chromosomes. More discussions on GA can be found in Goldberg (1989); and the use of GA in SVM has been illustrated in detail in Chen (2007), which interested readers can refer to.

Step 3: Testing

When GA2 is completed, the algorithm returns to GA1 and proceeds further. Each of the *N* chromosomes (binary codes) of the initial generation in GA1 is tested by its fitness function value, which is the MSE_p defined by (4.3). If the stopping criterion of GA1, which could be that the MSE_p is minimized or a predefined number of generations is reached, is satisfied, then the algorithm goes to Step 5; otherwise it goes to Step 4.

Step 4: Evolution

The chromosomes that have small fitness function values are selected and the crossover and mutation are conducted, thus an offspring generation is generated, and the algorithm goes back to Step 2.

Step 5: Global optimization of the binary code and model parameters

If the stopping criterion of GA1 is satisfied, then the algorithm is terminated, and the best binary code as well as the optimal values of SVM parameters is obtained. Then the proposed DDSRM can be used for prediction purpose, as shown in Figure 4.5 (*c*).

4.4 Numerical Examples

In this section we give two numerical examples, both of which are based on real data sets. These two data sets come from different application domains and have different failure data types, which could be helpful to validate the usefulness and generality of the proposed DDSRM and algorithm. We compare the performance of the proposed DDSRM with that of existing ones.

4.4.1 Example I

Consider software failure data used in Pham and Pham (2000), Tian and Noore (2005b), Su and Huang (2007), which are 22 inter-failure times taken from a telemetry network system by AT&T Bell Laboratories, shown in Table 4.2. In our experiment, we use the 19th to the 22nd inter-failure times as testing data, i.e., we set $d = 4$. Now we adopt the DDSRM in Figure 4.5, and we use the algorithm in Figure 4.4 to find the particular time lag terms that should be used and the optimal values of model parameters. Table 4.3 shows the results obtained.

Table 4.2 Software failure data taken from Pham and Pham (2000), Tian and Noore (2005b), Su and Huang (2007)

Failure number	Inter-failure time
1	5.50
2	1.83
3	2.75
4	70.89
5	3.94
6	14.98
7	3.47
8	9.96
9	11.39
10	19.88
11	7.81
12	14.59
13	11.42
14	18.94
15	65.30
16	0.04
17	125.67
18	82.69
19	0.45
20	31.61
21	129.31
22	47.60

Table 4.3 Model mining result and optimal values of parameters of SVM-based SRM with Gaussian kernel function, using software failure data in Table 4.2

The best binary code	C^*	σ^*	ε^*	MSE_p
00001100	2486	1.042	0.6417	670.56

It can be seen from Table 4.3 that by analyzing the data in Table 4.2, a failure x_i is found to be strongly correlated with two previous failures, x_{i-4} and x_{i-3}, thus these two time lag terms are used as model inputs for the SVM-based SRM; and the optimal values of model parameters, C^*, σ^*, and ε^*, are obtained as well. The resulting MSE_p is 670.56.

For comparative purpose, we examine the performance of existing SVM-based SRM which adopts equation (4.1), using the same data set in Table 4.2. To get the best model performance, we use the algorithm developed in Yang et al. (2007) to obtain the optimal values of model parameters. The results are $w^* = 5$, $C^* = 7120$, $\sigma^* = 1.1013$, and $\varepsilon^* = 5.9975$, under which $MSE_p = 2343.2$. It can be seen that the MSE_p is three more times bigger than that of the proposed DDSRM.

4.4.2 Example II

In the second example, we use the software failure data reported in Wood (1996), which are taken from a software release at Tandem Computers Company. This set of data is in the form of cumulative numbers of faults detected, shown in Table 4.4. The value of d is set to be the same as that in Example I, i.e., $d = 4$. Table 4.5 shows the results obtained using the proposed DDSRM in Figure 4.5 and the algorithm in Figure 4.4.

Table 4.4 Software failure data reported in Wood (1996)

Week	Cumulative faults detected
1	13
2	18
3	26
4	34
5	40
6	48
7	61
8	75
9	84
10	89
11	95
12	100
13	104
14	110
15	112
16	114
17	117
18	118
19	120

Table 4.5 Model mining result and optimal values of parameters of SVM-based SRM with Gaussian kernel function, using software failure data in Table 4.4

The best binary code	C^*	σ^*	ε^*	MSE_p
00001011	4159	0.298	0.5296	0.0487

In this example, it is found that a failure x_i is strongly correlated with three previous failures, x_{i-4}, x_{i-2}, and x_{i-1}, thus these three time lag terms are used as model inputs. The resulting MSE_p is 0.0487.

Similar as Example I, we use existing SVM-based SRM to analyze the software failure data in Table 4.4. For this example, the optimal values of model parameters are found to be $w^* = 1$, $C^* = 1131$, $\sigma^* = 0.3383$ and $\varepsilon^* = 0.0097$, under which $MSE_p = 1.42$. It can be seen that the MSE_p is once again much bigger than that of the proposed DDSRM.

The results obtained from the previous two examples verify that by using the proposed DDSRM with the developed hybrid GA-based algorithm, model performance could be significantly improved. This is actually expected because existing DDSRMs cannot cater for various failure correlations in a time series, e.g., for the data in Table 4.2, x_i is correlated with x_{i-4} and x_{i-3}; while for the data in Table 4.4, x_i is correlated with x_{i-4} , x_{i-2} , and x_{i-1} ; and hence the model performance would be affected.

As discussed, the proposed DDSRM is a generic model which includes the cases of existing DDSRMs. If by using the developed hybrid GA-based algorithm it is found that a failure is correlated with the most recent *w* failures, e.g., the best binary code is 00001111, then the proposed DDSRM reduces to an existing DDSRM.

4.5 Conclusion

In this chapter, we first point out a fundamental drawback of existing DDSRMs which seems to have affected the performance of existing models. Then we develop a generic DDSRM which can cater for various failure correlations in reality. Taking SVM-based SRM using Gaussian kernel function as an example, a hybrid GA-based algorithm is developed to discover the failure correlation and to obtain the optimal values of model parameters. Experimental results show that the proposed model outperforms existing DDSRMs.

The improvement of model performance is achieved at a cost of increase of required computational effort. In the proposed hybrid GA-based algorithm, GA1 is used to determine the best binary code which describes the specific failure correlation, and GA2 is used to determine the optimal model parameters. Compared with existing DDSRMs for which failure correlation is assumed to be a simple one, i.e., only consecutive failures are correlated, and only GA2 is used, it is expected that the time complexity of the proposed algorithm is greater. In our experiments on a normal personal computer, the hybrid GA-based algorithm could take up to forty minutes to be completed. This necessitates our future research on the improvement of the algorithm efficiency. More specifically, on one hand, we can adopt the use of more recent and advanced genetic algorithms such as the algorithm proposed in Ye et al. (2010). On the other hand, we can try incorporating priori information from the decision maker. This kind of information can refine the search of algorithm in a more reasonable and limited parameter space.

Chapter 5 Sensitivity Analysis of Release Time of Software Reliability Models Incorporating Testing Effort with Multiple Change Points

5.1 Basic Problem Description

Developing software reliability models is not the end of software reliability analysis. To guide management when to release the software based on these models is a typical application. In the following chapters, we will focus on another important aspect of software reliability analysis: release time determination.

Recently, incorporating testing effort into software reliability growth models (SRGMs) has received a lot of attention. The optimal release time problem considering testing effort was also discussed (Yamada et al., 1993; Huang and Kuo, 2002; Huang and Lyu, 2005a; Lin and Huang, 2008). However, most of the research assumes that parameters of the proposed models are known. In fact, there always exist estimation errors as parameters in testing effort function and SRGMs are generally estimated by least square estimation (LSE) method and maximum likelihood estimation (MLE) method respectively. It is necessary to conduct the sensitivity analysis to determine which parameter may have significant influence to the software release time. This is even more important when there are an increasing number of parameters involved in the model, such as the model proposed by Lin and Huang (2008).

Sensitivity analysis can be used to determine how sensitive the software release time is. It helps to find parameters that could significantly affect the solution to the release time. By showing how the software release time reacts against the changes in parameter values, the model is also evaluated and validated. In this chapter, sensitivity of the software release time is studied and different approaches are used, including one-factor-at-a-time approach, design of experiments (DOE) and global sensitivity analysis.

After the sensitivity analysis, significant parameters can be determined and they should be estimated precisely. However, this may not be possible due to the limited amount of information available. Thus, conservative estimation of release time is needed to avoid releasing the software too optimistically (Xie and Hong, 1998). To this end, interval estimation is recommended for use and the simulation results from global sensitivity analysis can just help in this.

The rest of this chapter is organized as follows. Section 5.2 introduces the general model incorporating testing effort and formulates the software release time problem. Section 5.3 discusses procedures when using different approaches to sensitivity analysis. In Section 5.4, an application example is given, and some interesting results are obtained. In Section 5.5, limitations of different approaches are highlighted. The interval estimation of optimal release time is discussed in Section 5.6 and it can be seen that results from global sensitivity analysis are very helpful in this. Concluding remarks are made in Section 5.7.

5.2 General Model Incorporating Testing Effort

To accurately model software failure process with SRGMs, incorporating testing effort has shown to be important and it has received a lot of attention. According to Lin and Huang (2008), multiple change points should be considered due to the changing testing efforts in reality. This model is adopted here as it is shown to be a general one with fairly accurate prediction capability (Lin and Huang, 2008). Specifically, with the consideration of arbitrary number of change points, the cumulative testing effort function is given by

$$W(t) = \begin{cases} \alpha\left(1 - \exp[-\beta_1 t^{\gamma_1}]\right), & 0 \le t \le \tau_1, \\ \alpha\begin{pmatrix} 1 - \exp[-\beta_1 \tau_1^{\gamma_1}] + \exp[-\beta_2 \tau_1^{\gamma_2}] \\ - \exp[-\beta_2 t^{\gamma_2}] \end{pmatrix}, & \tau_1 < t \le \tau_2, \\ \vdots & \vdots \\ \alpha\begin{pmatrix} \exp[-\beta_{m+1}\tau_m^{\gamma_{m+1}}] - \exp[-\beta_{m+1} t^{\gamma_{m+1}}] \\ + \sum_{k=1}^{m} \exp[-\beta_k \tau_{k-1}^{\gamma_k}] - \exp[-\beta_k \tau_k^{\gamma_k}] \end{pmatrix}, & \tau_m < t, \end{cases} \tag{5.1}$$

Based on the assumptions provided in Lin and Huang (2008), the mean value function representing the expected number of faults detected in time interval $(0, t]$ can be written as

$$m(t) = a\left[1 - e^{-rW(t)}\right]; \quad a > 0, 0 < r < 1, \tag{5.2}$$

where the boundary conditions are $m(0) = 0$ and $W(0) = 0$. Given (5.2), failure intensity can be also calculated by

$$\lambda(t) = \frac{dm(t)}{dt} = arw(t)e^{-rW(t)} \quad a > 0, 0 < r < 1, \tag{5.3}$$

In general, constructing a model is not the end. When the testing process proceeds, there will be fewer and fewer faults in the software. Accordingly, the software becomes more reliable. It is useful to provide information for management to decide when to stop the testing. Given a reliability target, the minimum testing time T required is generally calculated from the following formulation.

$$R(x \mid t) = \exp\left[-\{m(t+x) - m(t)\}\right] \tag{5.4}$$

$$R(x \mid t) \geq R_0 \tag{5.5}$$

In (5.4), $R(x \mid t)$ represents the conditional software reliability which is defined as the probability that the software will not fail given a specified time interval $(t, t+x]$ (Musa et al., 1987; Xie, 1991). By solving (5.4) and (5.5), the minimum testing time T required to achieve the given reliability target R_0 is received. However, the problem formulation above is under the testing reliability scenario. It means that the software is still under testing after release. In fact, operational reliability scenario is more reasonable as software codes will not be changed by customers after release (Yang and Xie, 2000). Therefore, from the customers' point of view, the operational reliability perspective is adopted here for further analysis. That is

$$R(x \mid t) = \exp\left[-\lambda(t)x\right] = \exp\left[-arw(t)e^{-rW(t)}x\right] \tag{5.6}$$

As there is no close form for the minimum testing time T to achieve a predetermined reliability target R_0 based on the model proposed by Lin and Huang (2008), numerical calculations are generally adopted. Without the loss of generality, x is set to 1 in $R(x \mid t)$ and R_0 is set to 0.95 in the following discussions.

5.3 Approaches to Sensitivity Analysis

In this section, sensitivity analysis of software release time formulated in the previous section is studied by various methods, i.e., one-factor-at-a-time approach, design of experiments (DOE) and global sensitivity analysis. The properties of each approach are also discussed.

5.3.1 One-Factor-at-a-Time Approach

One-factor-at-a-time approach is usually adopted due to its simplicity (Xie and Hong, 1998, Huang and Lyu, 2005b; Lo et al., 2005; Huang and Lo, 2006; Yang et al., 2008; Li et al., 2010). It is generally done by changing one parameter and setting the other parameters at their fixed values. It can be seen from (5.1) and (5.2) that there are totally (2m+5) model parameters to be investigated in our problem. When the parameter *a* is investigated to see how much the minimum testing time *T* is changed, T is in fact a function of *a* as other parameters are fixed using their estimated values. Then $S_{p,a}$ can be calculated, and it is defined as the relative change of the release time when *a* is changed by 100*p*% . That is

$$S_{p,a} = \frac{T(a+pa)-T(a)}{T(a)} \tag{5.7}$$

Similarly, $S_{p,r}$, $S_{p,\alpha}$, S_{p,β_i}, and S_{p,γ_i} $(i=1,2,\cdots,m+1)$ can be received in the same manner.

One-factor-at-a-time approach can help us to find the most sensitive parameter. For example, if $S_{p,a}$ is with the largest scale when p changes, then parameter a is regarded as the most sensitive parameter. Furthermore, it can also provide information about the trend of the release time with respect to each model parameter. By changing the value of *p*, we can check whether the minimum testing time required increases or decreases with respect to each model parameter.

5.3.2 Sensitivity Analysis through DOE

There are totally (2*m*+5) parameters of interest in the sensitivity analysis, i.e., a, r, α, β_i, and γ_i. It can be seen that one-factor-at-a-time approach is cumbersome when many parameters are involved. Thus, a more efficient way in conducting sensitivity analysis is required. DOE is an efficient approach and it is adopted by Xie et al. (2004). In the framework of DOE, the experiment can be explained as a test or series of tests where some purposeful changes are made to the input variables so that the reasons for the changes observed in the output response can be identified (Box et al., 1978; Montgomery and Runger, 1999). Specifically, in our problem, (2m+5) parameters (i.e., a, r, α, β_i, and γ_i) are input variables and the optimal release time T is the output response.

It is worth noting that there are totally (2m+5) parameters to be investigated and $2^{(2m+5)}$ runs are needed for a full factorial design. To improve the efficiency for conducting sensitivity analysis, a Resolution III fractional factorial design is adopted here and the interaction effects are assumed to be negligible. For more detailed process of Resolution III fractional factorial design, interested readers can refer to Box et al. (1978) and Montgomery and Runger (1999). In this part, a typical example of Resolution III fractional factorial design is shown in the following table and the use of it is briefly discussed.

Table 5.1 A saturated Resolution III fractional factorial design

θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	T
-	-	-	+	+	+	-	T_1
+	-	-	-	-	+	+	T_2
-	+	-	-	+	-	+	T_3
+	+	-	+	-	-	-	T_4
-	-	+	+	-	-	+	T_5
+	-	+	-	+	-	-	T_6
-	+	+	-	-	+	-	T_7
+	+	+	+	+	+	+	T_8

The experimental design shown in Table 5.1 is actually a saturated Resolution III factional factorial design, with which 7 factors with only 8 runs can be investigated. The sings '+' and '-' denote the high level and low level of each parameter respectively. In this design, suppose that the integration effects are negligible, the optimal release time can be estimated by using a linear model. That is

$$\begin{aligned}\hat{T} = E_0 &+ (E_1/2)x_1 + (E_2/2)x_2 + (E_3/2)x_3 \\ &+ (E_4/2)x_4 + (E_5/2)x_5 + (E_6/2)x_6 + (E_7/2)x_7\end{aligned} \tag{5.8}$$

where E_0 is the grand average and the other E_i's are the main effects of parameters. These terms can be calculated according to the following equations and significant parameters can be determined based on them. If parameter θ_{i} is the most significant parameter, then E_i will be with the largest absolute value.

$$\begin{aligned}
E_0 &= \frac{1}{8}(+T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7 + T_8) \\
E_1 &= \frac{1}{4}(-T_1 + T_2 - T_3 + T_4 - T_5 + T_6 - T_7 + T_8) \\
E_2 &= \frac{1}{4}(-T_1 - T_2 + T_3 + T_4 - T_5 - T_6 + T_7 + T_8) \\
E_3 &= \frac{1}{4}(-T_1 - T_2 - T_3 - T_4 + T_5 + T_6 + T_7 + T_8) \\
E_4 &= \frac{1}{4}(+T_1 - T_2 - T_3 + T_4 + T_5 - T_6 - T_7 + T_8) \\
E_5 &= \frac{1}{4}(+T_1 - T_2 + T_3 - T_4 - T_5 + T_6 - T_7 + T_8) \\
E_6 &= \frac{1}{4}(+T_1 + T_2 - T_3 - T_4 - T_5 - T_6 + T_7 + T_8) \\
E_7 &= \frac{1}{4}(-T_1 + T_2 + T_3 - T_4 + T_5 - T_6 - T_7 + T_8)
\end{aligned} \tag{5.9}$$

The design discussed above is actually of great importance in our study for the following two reasons: (1) if one change-point is used for describing the changeable testing effort function, i.e., m=1, there are just 7 parameters to be investigated; (2) Lin and Huang (2008) provided three numerical examples based on three real data sets, and all of them are with a single change-point, which may indicate that one change-point could be quite general in real applications.

In summary, sensitivity analysis through DOE can quickly identify the most sensitive parameter or a subset of input parameters which have the most significant influence on the solution. Compared with the other methods, this approach always enjoys the high efficiency. This is essentially in accordance with the original idea behind DOE, using the least resource to determine significant factors since the experiment could be very expensive or time consuming in the real world application.

5.3.3 Global Sensitivity Analysis

Global sensitivity analysis is widely discussed in recent years and it has drawn a lot of research attention (Sobol, 2001; Saltelli, 2002; Saltelli et al., 2008; Makowski 2006; Volkova et al., 2008; Benke et al., 2008; Yu and Harris, 2008). Compared with the previous two methods, where only special points or a local region of the parameter space is considered, the global sensitivity analysis can investigate the global parameter space and therefore the accuracy of the results can be improved (Saltelli et al., 2008; Yu and Harris, 2008). Specifically, it uses the additional knowledge we have about the model parameters, i.e., the distributions of model parameters.

The MLE method is commonly adopted for the estimation of parameters in SRGMs (Zhao and Xie, 1996; Wu et al., 2007). Also, it is theoretically sound and acceptable that the distributions of parameters are asymptotically normal. In our problem, parameters a and r are estimated in such a way according to Lin and Huang (2008).

While for the parameters α , β_i and γ_i in testing effort function, Lin and Huang (2008) adopted the LSE method for estimation. In this case, to construct reasonable distributions of these parameters, expert opinion is needed, and the triangular distribution is usually adopted (Park, 2007).

In general, there are several methods for making global sensitivity analysis. In this research, we restrict ourselves to the first-order Sobol indices with Monto Carlo simulation (Sobol, 2001; Saltelli, 2002; Saltelli et al., 2008). This is also reasonable as main effects of parameters are of more concern and first-order sensitivity indices just measure them. In our problem, $T = f(\theta_1, \theta_2, \cdots \theta_{2m+5})$ is the model under investigation where parameters $\theta_1, \theta_2, \cdots \theta_{2m+5}$ are input variables and the optimal release time T is the output response. The total variance of T used in global sensitivity analysis can be decomposed as follows:

$$V(T) = \sum_{i=1}^{2m+5} V_i + \sum_{1 \le i \le j \le 2m+5} V_{ij} + \cdots + V_{1,2,\cdots,2m+5} \tag{5.10}$$

where $V(T)$ is total variance of T induced by the (2m+5) parameters, $V_i = V[E(T|\theta_i)]$ measures the main effect of parameter θ_i and other terms measure the interaction effects. Given (5.10), the first-order sensitivity index is defined as

$$S_i = \frac{V_i}{V(T)} \tag{5.11}$$

The first–order sensitivity index given by (5.11) can measure the main effect of the parameter as well. Specifically, parameter θ_i has a significant influence on the solution of software release time when S_i is close to one. By contrast, θ_i is not a sensitive parameter when S_i is close to zero. Moreover, interaction effects in the model can be measured with $1-\sum_i S_i$.

To calculate S_i for each parameter, V_i and $V(T)$ are required to be estimated in advance. These terms can be computed by using Monte Carlo simulation method discussed in Saltelli (2002) and Saltelli et al. (2008), where the input parameters θ_i's are assumed uncorrelated with one another. In fact, Saltelli (2002) has justified the effectiveness and efficiency of the algorithm theoretically, and interested readers can refer to it for detailed discussions. On the other hand, the independence assumption for parameters is commonly adopted when global sensitivity analysis is used (Saltelli et al., 2008; Makowski 2006; Volkova et al., 2008). According to Saltelli et al. (2008), the reason for this assumption lies in the fact that dependent input samples are more difficult to generate. More seriously, the sample size needed to compute sensitivity measures for non-independent samples is much larger compared with the case of uncorrelated samples.

The principle of the algorithm is to generate random samples of parameters according to their distributions. The base parameter values are quasi-random numbers generated using the Latin hypercube sampling method (Helton and Davis, 2003). Suppose that θ_i is with normal distribution, 2*N* random numbers of θ_i are generated according to this information. The first *N* of them are in matrix *A* denoted by $\theta_i^{(1)}$, $\theta_i^{(2)}$,…, $\theta_i^{(N)}$.

While the rest *N* of them are put into matrix *B*, where they are denoted by $\theta_{2m+5+i}^{(1)}$, $\theta_{2m+5+i}^{(2)}, \ldots, \theta_{2m+5+i}^{(2)}$. Specifically, the matrix *A* and matrix *B* are given by

$$A = \begin{bmatrix} \theta_1^{(1)} & \theta_2^{(1)} & \cdots & \theta_i^{(1)} & \cdots & \theta_{2m+5}^{(1)} \\ \theta_1^{(2)} & \theta_2^{(2)} & \cdots & \theta_i^{(2)} & \cdots & \theta_{2m+5}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \theta_1^{(N-1)} & \theta_2^{(N-1)} & \cdots & \theta_i^{(N-1)} & \cdots & \theta_{2m+5}^{(N-1)} \\ \theta_1^{(N)} & \theta_2^{(N)} & \cdots & \theta_i^{(N)} & \cdots & \theta_{2m+5}^{(N)} \end{bmatrix} \tag{5.12}$$

and

$$B = \begin{bmatrix} \theta_{2m+6}^{(1)} & \theta_{2m+7}^{(1)} & \cdots & \theta_{2m+5+i}^{(1)} & \cdots & \theta_{4m+10}^{(1)} \\ \theta_{2m+6}^{(2)} & \theta_{2m+7}^{(2)} & \cdots & \theta_{2m+5+i}^{(2)} & \cdots & \theta_{4m+10}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \theta_{2m+6}^{(N-1)} & \theta_{2m+7}^{(N-1)} & \cdots & \theta_{2m+5+i}^{(N-1)} & \cdots & \theta_{4m+10}^{(N-1)} \\ \theta_{2m+6}^{(N)} & \theta_{2m+7}^{(N)} & \cdots & \theta_{2m+5+i}^{(N)} & \cdots & \theta_{4m+10}^{(N)} \end{bmatrix}. \tag{5.13}$$

With *A* and *B*, a new matrix C_i is received by substituting the *i*th column from *A* into *B* and it is

$$C_i = \begin{bmatrix} \theta_{2m+6}^{(1)} & \theta_{2m+7}^{(1)} & \cdots & \theta_i^{(1)} & \cdots & \theta_{4m+10}^{(1)} \\ \theta_{2m+6}^{(2)} & \theta_{2m+7}^{(2)} & \cdots & \theta_i^{(2)} & \cdots & \theta_{4m+10}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \theta_{2m+6}^{(N-1)} & \theta_{2m+7}^{(N-1)} & \cdots & \theta_i^{(N-1)} & \cdots & \theta_{4m+10}^{(N-1)} \\ \theta_{2m+6}^{(N)} & \theta_{2m+7}^{(N)} & \cdots & \theta_i^{(N)} & \cdots & \theta_{4m+10}^{(N)} \end{bmatrix} \tag{5.14}$$

Finally, we calculate the software release time for all the parameter values in the matrices *A*, *B*, and C_i, resulting three $N \times 1$ vectors:

$$T_A = f(A) \quad T_B = f(B) \quad T_{C_i} = f(C_i) \tag{5.15}$$

The V_i and $V(T)$ in (5.11) are then obtained by

$$V_i = T_A \cdot T_{C_i} - f_0^2 = (1/N)\sum_{j=1}^{N} T_A^{(j)} T_{C_i}^{(j)} - f_0{}^2 \tag{5.16}$$

and

$$V(T) = T_A \cdot T_A - f_0^2 = (1/N)\sum_{j=1}^{N} \left[T_A^{(j)}\right]^2 - f_0{}^2 \tag{5.17}$$

where f_0 is the mean of T_A given by

$$f_0 = (1/N)\sum_{j=1}^{N} T_A^{(j)} \tag{5.18}$$

The above mentioned algorithm can be explained in a ‘hand waving’ fashion as illustrated in Saltelli et al. (2008). In the scalar product $T_A \cdot T_{C_i}$ given by (5.16), values of *T* computed from matrix *A* are multiplied by those computed from matrix C_i where all parameters but θ_i are resampled. Then, high and low values of T_A and

T_{C_i} are randomly associated if θ_i is non-influential. On the contrary, if θ_i is influential, high (or low) values of T_A and T_{C_i} will be preferentially multiplied.

5.4 An Illustrative Example

To illustrate the application of various sensitivity methods in our problem, the third data set used in Lin and Huang (2008) is adopted here. The data set actually origins from software release one in Wood (1996) from Tandem Computers Company. As the weekly testing effort consumption gradually decreased from the 11th week, τ_1 is set to 11 and m is equal to one. Parameters are estimated in two ways according to Lin and Huang (2008): LSE method is used for estimating the parameters in test effort function and MLE method for the parameters a and r in SRGMs. Estimators of parameters are received in the same fashion in the following sensitivity analysis. The numerical results will help us to further understand the use of each approach.

5.4.1 Results from One-Factor-at-a-Time Approach

Following the procedures discussed in Section 5.3.1, some numerical results from the one-factor-at-a-time approach are shown in Table 5.2.

Table 5.2 Some numerical results from one-factor-at-a-time approach

p	-30%	-20%	-10%	10%	20%	30%
$S_{p,a}$	-0.097	-0.061	-0.029	0.027	0.051	0.074
$S_{p,r}$	0.111	0.077	0.040	-0.041	-0.083	-0.126
$S_{p,\alpha}$	0.111	0.077	0.040	-0.041	-0.083	-0.126
S_{p,β_1}	0.078	0.050	0.025	-0.023	-0.045	-0.065
S_{p,β_2}	0.183	0.112	0.052	-0.046	-0.087	-0.125
S_{p,γ_1}	0.180	0.133	0.073	-0.084	-0.173	-0.259
S_{p,γ_2}	0.850	0.517	0.232	-0.186	-0.330	-0.445

From the table, it can be seen that the shape parameter γ_2 in the testing effort function is the most sensitive parameter with the largest scale of S_{p,γ_2}. Moreover, optimal release time *T* is decreasing with the increase of r, α, β_1, β_2, γ_1, and γ_2 respectively. It is only increasing with the increase of the parameter a. Accordingly, overestimation of r, α, β_1, β_2, γ_1, γ_2 and underestimation of a, which implies a underestimation of release time should be avoided. Because it will be costly by making consumers experience more failures when the software is released too early.

It should be also noted that the *S* values in second row and third row are the same. The same values are received as the reliability function has the same amount of change when parameters r and α change and optimal release time is uniquely determined by the reliability function. Specifically, let *x* equal to 1 and equation (5.6) has the same form when r and α change by 100*p*%. That is

$$R(1\,|\,t) = \exp\left[-\lambda(t)\right] = \exp\left[-ar(1+p)w(t)e^{-r(1+p)W(t)}\right]. \tag{5.19}$$

7.4.2 Results from Sensitivity Analysis through DOE

Based on the same data set, it can be calculated that the respective relative changing rates of the maximum likelihood estimators $\hat{a}$ and $\hat{r}$ are approximately

$$\frac{\sqrt{Var(\hat{a})}}{\hat{a}} = 1\%, \qquad \frac{\sqrt{Var(\hat{r})}}{\hat{r}} = 16.9\%,$$

which does not show significant difference. Therefore, according to Xie et al. (2004), factor levels for parameters can be described as follows, and they are consistently used for the other parameters.

-: Decrease by 30% of original value

+: Increase by 30% of original value

The Resolution III fractional factorial design and analytical results are shown in Table 5.3, Table 5.4 and depicted in Figure 5.1 in a descending manner.

Table 5.3 Fractional factorial design

a	r	α	β_1	β_2	γ_1	γ_2	T
-	-	-	+	+	+	-	81.51
+	-	-	-	-	+	+	43.16
-	+	-	-	+	-	+	30.92
+	+	-	+	-	-	-	205.30
-	-	+	+	-	-	+	41.99
+	-	+	-	+	-	-	163.97
-	+	+	-	-	+	-	35.35
+	+	+	+	+	+	+	19.19

Table 5.4 Main effects of parameters

(1-a ; 2-r ; 3-α ; 4-β_1 ; 5-β_2 ; 6-γ_1 ; 7-γ_2)

E_1	E_2	E_3	E_4	E_5	E_6	E_7
60.46	-9.97	-25.10	18.65	-7.55	-65.74	-87.72

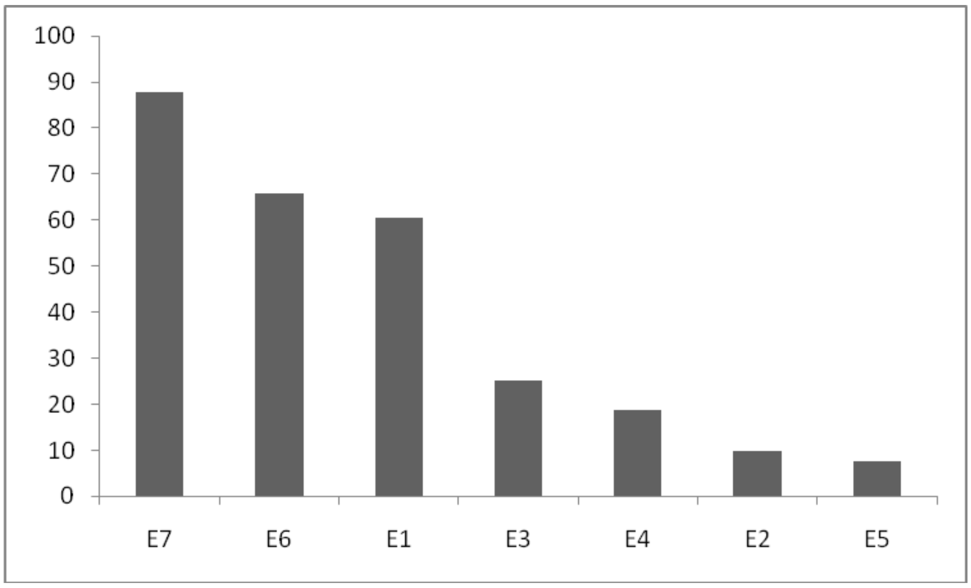


Figure 5.1 Main effects of parameters (absolute value)

(1-a ; 2-r ; 3-α ; 4-β_1 ; 5-β_2 ; 6-γ_1 ; 7-γ_2)

It can be easily seen that parameter γ_2 is the most dominant factor which affects the solution to software release time; parameters γ_1 and a have the second significant influence on the solution followed by α , β_1 , r and β_2 . The significance of parameters γ_2 and γ_1 is reasonable when we come back to see equations given by (5.1) and (5.6). It can be seen that these two parameters are the shape parameters in Weibull testing effort function in (5.1) and the Weibull testing effort function is an exponent in the second exponential function of (5.6). However, the significance of parameter a is quite questionable since it is the only parameter, which is not related to the exponent in the second exponential function of (5.6). Furthermore, it is not in accordance with the results shown in Xie and Hong (1998) and Xie et al. (2004) that the expected number of faults a is generally less sensitive than the failure detection rate r. This inaccurate result could be caused by the assumptions used in the design, i.e., the use of the linear model, the arbitrariness of factor level labels and the ignorance of interaction.

5.4.3 Results from Global Sensitivity Analysis

In global sensitivity analysis, the base sample N should be predetermined. We set N equal to 200000 in our application example as a large number of N can produce stable estimates of the first-order sensitivity indices with low variability (Makowski et al., 2006). Further, since parameters α, β_1, β_2, γ_1, and γ_2 in the testing effort function are estimated by LSE method, expert opinion is needed to construct reasonable distributions of them. In this case, the triangular distribution is generally adopted (Park, 2007). Due to this consideration, for parameters in the testing effort function, suppose that their most probable values are the estimated values from LSE method; their highest and lowest values are 30% increase and decrease of the most probable values respectively. However, it should be noted that this assumption is adopted here for illustrative purpose. In real applications, distributions of these parameters can be different from each other considering their different physical meanings in the testing effort function. While, for parameters a and r estimated by MLE method, it is theoretically sound and acceptable that these parameters are normally distributed.

Following the standard procedures of global sensitivity analysis discussed in Section 5.3.3, results of the first-order sensitivity indices are listed in Table 5.5 and depicted in Figure 5.2 in a descending manner.

Table 5.5 Results of the first-order sensitivity indices

(1-a ; 2-r ; 3-α ; 4-β_1 ; 5-β_2 ; 6-γ_1 ; 7-γ_2)

S_1	S_2	S_3	S_4	S_5	S_6	S_7
0.0095	0.0664	0.0398	0.017	0.0497	0.1125	0.6951

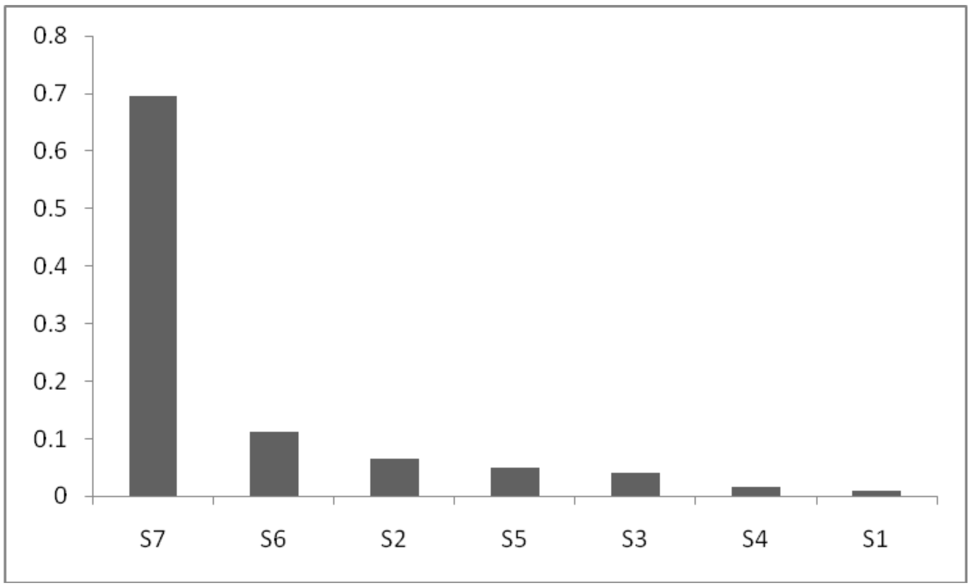


Figure 5.2 Results of first-order sensitivity indices in a descending manner

(1-a ; 2-r ; 3-α ; 4-β_1 ; 5-β_2 ; 6-γ_1 ; 7-γ_2)

For the determination of software release time, it can be seen that parameter γ_2 is the most sensitive parameter. Its value of first-order sensitivity index is equal to 0.6951. It means that 69.51% of the software release time variance would be left if the parameter γ_2 is undetermined. According to Figure 5.2, the other parameter γ_1 also has a significant influence on the determination of software release time. Its value of first-order sensitivity index is equal to 0.1125. The significance of these two parameters can also be explained as before. For the other 5 parameters, they are not significant according to the Pareto principle as parameters γ_1 and γ_2 has accounted for roughly 80% (80.76% exactly) of the software release time variance.

Furthermore, the parameter *a*, which is determined as a significant parameter in DOE, is determined as the most insignificant parameter in global sensitivity analysis. The result here is in accordance with the intuition of equation (5.6). Therefore, it provides further evidence that global sensitivity analysis could be more accurate. In addition, since $1-\sum_i S_i = 1\%$, it indicates that the interaction effects are negligible in this application example.

5.5 Limitations of Different Approaches

In our study, different approaches to sensitivity analysis are with different advantages. One-factor-at-a-time approach is simple and straightforward; DOE is with high efficiency utilizing the least amount of resources and global sensitivity analysis can investigate the global parameter space to get more accurate results. However, at the same time, the limitations of different approaches need to be highlighted here and special care should be taken for them.

For one-factor-at-a-time approach, the most restrictive assumption is that all effects of parameters are independently estimated. Thus, it only focuses on some special points of the parameter space and fails to investigate these parameters simultaneously. Furthermore, when there are a large number of parameters to be investigated, large number of observations are needed.

As to DOE, it is generally based on the linear model, which could be the most restrictive assumption, since in our study the relationship between the optimal release time and input parameters is complex and nonlinear. Probably because of this, in the application example, DOE treats the parameter a as a third significant parameter wrongly. Possible ways to solve this kind of problem could be the use of 3 level experimental design and nonlinearity check by statistical tests. However, the use of them will greatly increase the complexity of the method, which may not be desirable.

Additionally, interactions are assumed to be negligible in our Resolution III fractional factorial design. Compared with the previous assumption, this one could be a minor one. On one hand, in our application example, the results from global sensitivity analysis indicate that the interaction effects can be ignored. On the other hand, the results of DOE are generally not affected when the interaction effects are assumed to be negligible according to Taguchi et al. (2005).

As to global sensitivity analysis, the most restrictive assumption is that the priori knowledge about the distributions of parameters is needed. It is known that this kind of information may not be available all the time. Some common ways to solve this kind of problem is to use some simple distributions based on the expert opinion, i.e., the uniform distribution or the triangular distribution (Makowski, 2006; Saltelli et al., 2008). However, the impact of the accuracy of these estimations on the final results of sensitivity measures still needs to be investigated as an inaccurate estimation of the distribution can probably lead misleading results. Future research on this problem is needed.

Additionally, compared with the previous two methods, global sensitivity analysis is more computationally expensive as shown in Table 5.6. The numbers in the table are received in the following manner. There are totally ($2m$+5) parameters to be investigated. In one-factor-at-a-time (OFAT) approach, -30% to 30% is selected and 10% is set as the step, thus, $6(2m+5)$ T values are needed; in DOE, 2^k runs of Resolution III design can investigate 2^k-1 factors (Montgomery and Runger, 1999), therefore, it should be greater than or equal to $2m$+6 and less than $4m$+12; in global sensitivity analysis (GSA), according to the simulation procedures discussed in this chapter, $N(2m+7)$ T values are needed. Although global sensitivity analysis is the most computationally expensive method, this limitation may not be that serious in practice. The reason lies in the fact that values of optimal release time are just numerical calculations rather than costly and time-consuming real world experiments.

Table 5.6 Comparison of computation resources needed

Approaches	No. of optimal release time values to be calculated
OFAT	$6(2m+5)$
DOE	2^k ($2m+6\leq 2^k<4m+12$, k is a positive integer)
GSA	$N(2m+7)$

5.6 Interval Estimation from Global Sensitivity Analysis

Significant parameters can be determined after sensitivity analysis. Usually, for these parameters, they should be estimated more precisely. One possible way to do this is to gather some information from similar projects as shown in Xie et al. (1999). However, parameters are unknown in nature, and they are estimated based on the limited amount of data. The point estimate of optimal software release time could be too optimistic. In fact, the optimal release time is a random variable with the consideration of the estimation errors in parameters. It is necessary to provide management with more confidence with respect to the estimation of the release time. With the consideration of this, interval estimation is recommended for use (Zhao and Xie, 1993).

Previously, the interval estimation is usually received by standard method in statistical analysis where large sample properties of the MLE are adopted (Zhao and Xie, 1993). However, such analysis is not applicable in our analysis since parameters in testing effort function are estimated by LSE method. In this case, an alternative way is needed and simulation results from global sensitivity analysis can just help in this. Since there are totally $N(2m+7)$ values of optimal release time T based on the simulation results from global sensitivity analysis, where $N(2m+7)$ is large enough (i.e., in our application example it is equal to 1800000), then cumulative distribution function (CDF) of T can be estimated and it actually has all information we need for uncertainty analysis (Helton and Davis, 2003). Specifically, for any predetermined value denoted by x, the cumulative distribution function can be estimated in such a way that

$$P\{T < x\} = \sum_{i=1}^{N(2m+7)} \delta_T(T_i) \frac{1}{N(2m+7)} \tag{5.20}$$

where

$$\delta_T(T_i) = \begin{cases} 1 & T_i < x \\ 0 & T_i \geq x \end{cases} \tag{5.21}$$

Suppose that the confidence level is θ, then the lower bound and upper bound of the optimal release time *T* can be obtained easily according to

$$P\{T \leq T_L\} = \frac{\theta}{2} \text{ and } P\{T \geq T_U\} = \frac{\theta}{2} \tag{5.22}$$

It means that the true value of the optimal release time will be included in the interval estimate [T_L T_U] with probability (1-θ). Compared with the point estimate of the optimal release time, the interval estimation is generally more robust and informative (Zhao and Xie, 1993). The length of the interval estimate calculated by T_U-T_L can measure the precision of the estimation of the optimal release time. Specifically, a narrow confidence interval indicates the high accuracy. In practice, under a prescribed length *L*, the testing process can be controlled by the confidence interval. If T_U-T_L is less than or equal to *L*, the precision is supposed to be acceptable; otherwise, further testing is required to improve the precision (Zhao and Xie, 1993). However, it is possible that the predetermined threshold cannot be satisfied due to the time constraint or the available cost budget.

For illustration, based on the same data set, the 90% confidence interval is calculated and it is given by [38.62 102.84]. It can be seen that the length of the interval is 64.22. Therefore, if L<64.22, the testing process is required to be continued if the time constraint and the cost budget are not exceeded. It is worth noting that 90% confidence interval is used here. However, the selection of the confidence level could be quite different from company to company, and from project to project. Since different confidence intervals are just simple calculations, the results of them are omitted here.

5.7 Conclusion

In this chapter, different approaches to sensitivity analysis are adopted and properties of them are discussed. Especially, the assumptions are highlighted which can help practitioners better understand the limitations that need attention in the real application. Results from traditional methods like the one-factor-at-a-time approach and DOE may not be accurate enough. Thus, global sensitivity analysis is recommended for use due to the consideration of the global parameter space. Furthermore, global sensitivity analysis possesses another advantage that other methods do not have. Results from it not only help to determine the sensitive parameters, but also provide further information for management to decide when to release software under parameter uncertainty. With the use of the interval estimation for the optimal release time, the precision of the estimation can be measured and controlled.

Chapter 6 A Risk-Based Approach for Software Release Time Determination with Delay Costs Considerations

For software release time determination problem, meeting the reliability requirement is of great importance. This is because customers generally have a minimum reliability requirement, and it can be specified in the contract. In order to check whether the reliability requirement is satisfied, software reliability model is generally adopted to predict the reliability of software. Most existing research on release time determination assumes that parameters in the software reliability model are known and the reliability estimate is accurate (Okumoto and Goel, 1980; Yamada and Osaki, 1985; Xie and Yang, 2003; Boland and Chuiv, 2007; Huang and Lyu, 2005a; Ho et al., 2008; Liu and Chang, 2007; Yang et al., 2008). In practice, however, there exists the risk that the reliability requirement cannot be guaranteed due to parameter uncertainty, and such risk can be as high as 50% when the mean value is used, as shown in this chapter. It is necessary for management to reduce this risk to a lower level, and software is expected to be tested longer. The challenging point is that this will inevitably increase costs of the testing process. In order to balance between reducing the risk and controlling the penalty cost associated with it, in this chapter, we develop a new decision model for software release time determination, and apply multi-attribute utility theory (MAUT) to optimize risk and cost simultaneously.

The rest of this chapter is organized as follows. Section 6.1 introduces the general approach of quantifying the uncertainty of model parameters. In Section 6.2, limitations of existing research on software release time determination are further discussed, which motivate us to incorporate the risk that software cannot meet the reliability requirement into consideration. In addition, attributes including risk and penalty cost are formulated. In Section 6.3, the decision model based on MAUT is developed, and the procedure on how to construct it is discussed in detail. In Section 6.4, an application example is provided for illustrative purpose. In Section 6.5, a simplification of the decision model is introduced with some non-restrictive assumptions. With the simplified decision model, analytical tractability is possessed and the complexity of the decision process is greatly reduced. In Section 6.6, threats to validity are discussed. Finally, concluding remarks are made in Section 6.7

6.1 Quantifying Parameter Uncertainty

Model parameters have to be estimated based on the recorded failure data. A common method is to adopt the maximum likelihood estimation (MLE) technique (Zhao and Xie, 1996; Wu et al., 2007). Using the MLE approach allows parameter uncertainty to be quantified in terms of variability.

Suppose that there are totally m model parameters to be estimated denoted by $\theta_1, \theta_2, \cdots, \theta_m$. Let n_i denote the number of failures observed in the time interval $[t_{i-1}, t_i)$, where $0 = t_0 < t_1 < \cdots < t_k = t$ and t is the time at which the testing process has experienced. The likelihood function for a non-homogeneous Poisson process (NHPP) model with mean value function $m(t)$ is

$$L = \prod_{i=1}^{k} \frac{\left[m(t_i) - m(t_{i-1})\right]^{n_i} \exp\left\{-\left[m(t_i) - m(t_{i-1})\right]\right\}}{n_i!} \tag{6.1}$$

It is worth noting here that the mean value function $m(t)$ contains the m model parameters $\theta_1, \theta_2, \cdots, \theta_m$. Point estimates of the model parameters can be determined by maximizing the likelihood function above. To quantify the parameter uncertainty, the variances of the estimators of the parameters can be calculated following the asymptotic theory for MLE (Nelson, 1982). Specifically, the Fisher information matrix can be calculated as

$$I(\theta_1, \cdots, \theta_m) = \begin{bmatrix} -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_1^2}\right] & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_2 \partial \theta_1}\right] & \cdots & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_m \partial \theta_1}\right] \\ -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_1 \partial \theta_2}\right] & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_2^2}\right] & \cdots & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_m \partial \theta_2}\right] \\ \cdots & \cdots & \cdots & \cdots \\ -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_1 \partial \theta_m}\right] & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_2 \partial \theta_m}\right] & \cdots & -E\left[\dfrac{\partial^2 \ln L}{\partial \theta_m^2}\right] \end{bmatrix} \tag{6.2}$$

According to the standard theory of MLE, when the data size is large, $\left[\theta_1, \theta_2 \cdots, \theta_m\right]$ converges to m-variate normal distribution with mean $[\hat{\theta}_1, \hat{\theta}_2 \cdots, \hat{\theta}_m]$ and variance $[Var(\hat{\theta}_1), Var(\hat{\theta}_2), \cdots, Var(\hat{\theta}_m)]$. The asymptotic covariance matrix which is the inverse of the Fisher information matrix is given by

$$V = I^{-1} = \begin{bmatrix} Var(\hat{\theta}_1) & Cov(\hat{\theta}_1,\hat{\theta}_2) & \cdots & Cov(\hat{\theta}_1,\hat{\theta}_m) \\ Cov(\hat{\theta}_2,\hat{\theta}_1) & Var(\hat{\theta}_2) & \cdots & Cov(\hat{\theta}_2,\hat{\theta}_m) \\ \cdots & \cdots & \cdots & \cdots \\ Cov(\hat{\theta}_m,\hat{\theta}_1) & Cov(\hat{\theta}_m,\hat{\theta}_2) & \cdots & Var(\hat{\theta}_m) \end{bmatrix} \tag{6.3}$$

The two sided approximate $100\alpha\%$ confidence interval for model parameter is

$$\theta_U = \hat{\theta} + Z_{\alpha/2}\sqrt{Var(\hat{\theta})} \quad \text{and} \quad \theta_L = \hat{\theta} - Z_{\alpha/2}\sqrt{Var(\hat{\theta})} \tag{6.4}$$

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

Moreover, based on the covariance matrix, the uncertainty of other quantities which are functions of parameters $[\theta_1, \theta_2 \cdots, \theta_m]$ can also be quantified. For example, let $f = f(\theta_1, \theta_2 \cdots, \theta_m)$ represent the quantity of interest and $\hat{f} = f(\hat{\theta}_1, \hat{\theta}_2 \cdots, \hat{\theta}_m)$ be the estimate. The variance for the $\hat{f}$ is estimated as

$$Var(\hat{f}) = \sum_{i=1}^{m}\left(\frac{\partial f}{\partial \theta_i}\right)^2 Var(\hat{\theta}_i) + \sum_{i=1}^{m}\sum_{\substack{j=1 \\ i\neq j}}^{m}\left(\frac{\partial f}{\partial \theta_i}\right)\left(\frac{\partial f}{\partial \theta_j}\right) Cov(\hat{\theta}_i, \hat{\theta}_j) \tag{6.5}$$

where $\partial f / \partial \theta_i$ is evaluated at $[\hat{\theta}_1, \hat{\theta}_2 \cdots, \hat{\theta}_m]$. The two sided approximate $100\alpha\%$ confidence interval for f is

$$f_U = \hat{f} + Z_{\alpha/2}\sqrt{Var(\hat{f})} \quad \text{and} \quad f_L = \hat{f} - Z_{\alpha/2}\sqrt{Var(\hat{f})} \tag{6.6}$$

A confidence interval for the parameter is a measure of the parameter uncertainty. In the following sections, attention will be focused on issues related to the software release time problem, which is an important decision that has to be made by managers in software development companies. In fact, the optimal software release time given the reliability requirement can be treated as a function of model parameters, and the uncertainty of it can be quantified based on the discussions above.

6.2 Model Formulation

Considering the software reliability requirement aspect for software release time determination, with the software reliability model and a specified minimum reliability level R_0, the decision problem is typically formulated as

$$R(x \mid t) \geq R_0, \tag{6.7}$$

where $R(x \mid t)$ is the conditional software reliability, which is defined as the probability that the software will not fail within a specified time interval $(t, t+x]$. The optimal release time T is then the minimum testing time required to satisfy this reliability target R_0. In most software reliability models, there are a set of parameters $\theta_1, \theta_2, \cdots, \theta_m$. Then the optimal release time T can then be represented by $T = f(\theta_1, \theta_2, ..., \theta_m)$, where f denotes the mapping function. In solving for the optimal release time, most existing research assumes that these model parameters are known with certainty, and $R(x \mid t)$ can model exactly the actual software reliability (Okumoto and Goel, 1980; Yamada and Osaki, 1985; Xie and Yang, 2003; Boland

and Chuiv, 2007; Huang and Lyu, 2005a; Ho et al., 2008; Liu and Chang, 2007; Yang et al., 2008).

6.2.1 Risk Considerations

In reality, however, exact values for these model parameters are unknown. These parameters are estimated based on the observed test data. Parameter uncertainty arises since the estimated parameters are subject to the random variations in the data (Dai et al., 2007). With parameters estimated from observed data, the software reliability computed from these models is no longer exact. Therefore, the optimal release time T given a reliability target is no longer a fixed value but a random variable. When the model parameters are estimated by the MLE method, based on the standard statistical analysis (Nelson, 1982), the optimal release time T given a reliability target is asymptotically normally distributed with mean $\hat{T}$ and variance $Var(\hat{T})$ as discussed in the previous section. Here, $\hat{T}$ is the release time given the reliability target R_0 obtained from solving (6.7) with the estimated parameters, and $Var(\hat{T})$ is the variance of $\hat{T}$. Detailed discussions on these results are shown in the previous section, and Figure 6.1 illustrates the uncertainty in the optimal release time T with $\hat{T} = 30$ and $Var(\hat{T}) = 25$.

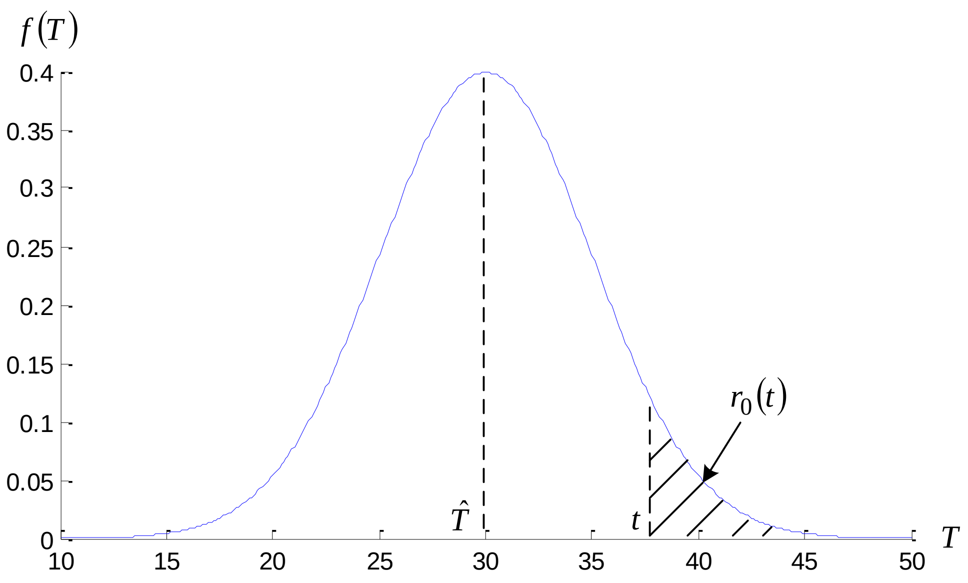


Figure 6.1 An illustrative example of the distribution of the optimal release time *T* given a reliability requirement

Based on this additional uncertainty in the optimal decision *T*, the risk that software cannot meet the reliability requirement when it is released at time *t* can be quantified as

$$P(R(x \mid t) < R_0) = r_0(t) = 1 - \int_{-\infty}^{\frac{t-\hat{T}}{\sqrt{Var(\hat{T})}}} \varphi(x)dx, \tag{6.8}$$

where $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ is the probability density function of standard normal distribution. In Figure 6.1, $r_0(t)$ is the area of the shaded region. It can be seen that when the mean value of release time $\hat{T}$ is used, there is 50% chance that the reliability requirement cannot be guaranteed. Since the reliability requirement is the software vendor’s commitment and it is generally specified in the contract, such risk can be too high to be acceptable. As a result, reducing the risk to a lower level to improve the confidence of the software quality becomes to an important issue. With the consideration of this, the risk-based release time T_R is recommended and it is given by

$$T_R = \hat{T} + Z_{r_0}\sqrt{Var\left(\hat{T}\right)}, \tag{6.9}$$

where r_0 denotes the acceptable risk level from management and Z_{r_0} is the (1-r_0) quantile of the standard normal distribution. As seen from (6.9), the use of risk-based release time requires a delay of release, increasing the testing time by $Z_{r_0}\sqrt{Var\left(\hat{T}\right)}$. This increased testing (and hence delay) increases the costs of the testing process. This is a useful approach if the developers and management are certain of the risk level required and are committed to achieve it at all costs. More often than not, it is easier to elicit a maximum tolerable risk value (although preference may be to drive it to zero), and software projects have to work within a budget.

6.2.2 Cost Considerations

From the management's perspective, it is also important to control the penalty cost due to the use of risk-based release time. Based on the generalized software cost model proposed by Pham and Zhang (1999), such penalty cost is the additional general testing cost (e.g., the salaries to be paid for testing team members) and the additional expected fault removal cost during the testing process. Specifically, the expected general testing cost $C_1(t)$ and the expected cost to remove errors during testing phase $C_2(t)$ are given by

$$C_1(t) = c_1 t^{\kappa},\ C_2(t) = c_2 m(t) \mu_y, \tag{6.10}$$

where c_1 is the software test cost per unit time, κ is the discount rate of the testing cost due to the learning effect, c_2 is the cost of removing an error per unit time during the testing phase and μ_y is the expected time of removing an error during this period. It is worth noting that risk is expected to be less than 50% from management's point of view. Therefore, we have $t \in [\hat{T}, +\infty)$. Accordingly, the penalty cost at the time *t* is obtained as

$$C_p(t) = c_1\left[(t)^{\kappa} - \left(\hat{T}\right)^{\kappa}\right] + c_2\left[m(t) - m\left(\hat{T}\right)\right]\mu_y. \tag{6.11}$$

In summary, the discussions above indicate that reducing the risk and controlling the penalty cost are two important criteria that should be considered simultaneously when determining the software release time. In this decision process, these two objectives contradict each other because the use of risk-based release time can inevitably increase the testing costs. In this case, it is necessary to incorporate management's preference into the decision process to make a compromise between these two criteria. To the best of our knowledge, these issues have not been highlighted and studied in the literature. In order to resolve these difficulties, multi-attribute utility theory (MAUT) is adopted, and a decision model is developed based on it for the determination of optimal risk-based release time. The proposed decision model can help management have a broader view of the software release time determination problem.

6.3 The Decision Model Based on MAUT

In MAUT, some independence assumptions, such as preferential independence, utility independence and additive independence, are used for a more practical form of the multi-utility function. It is worth noting that these assumptions are commonly accepted in practice. Moreover, it has been shown that even when these assumptions are violated, the additive multi-attribute function can provide fairly good approximations (Edwards, 1977; Farmer, 1987). For more detailed discussions on the multi-attribute function when independence assumptions are not held, interested readers can refer to (Keeney and Raiffa, 1976). In this thesis, we will adopt these commonly used assumptions.

The application of MAUT is based on a one-dimensional multi-attribute utility function, which is the measure of the attractiveness of the conjoint outcome of attributes given a specified alternative. The additive form of the multi-attribute utility function is given by

$$U(d_1, d_2, \ldots d_n) = \sum_{i=1}^{n} w_i u(d_i), \tag{6.12}$$

where each attribute is denoted by d_i, i=1,2,…n, the attractiveness of each attribute is represented by the single utility function $u(d_i)$ and w_i's are the scaling constants allocated for different single utility functions. The scaling constants represent the different importance weights for the utilities of attributes, and the sum of them is equal to 1 (von Winterfeldt and Edwards, 1986). By maximizing the multi-attribute utility function, the best alternative is obtained, under which the attractiveness of the conjoint outcome of attributes is optimized.

The main reason for the selection of MAUT in our problem is that typical management's scenarios can be appropriately represented within the structure of it. In our problem, there are two conflicting criteria to be balanced for the determination of optimal risk-based release time: minimizing the risk and minimizing the penalty cost. Hence, risk and penalty cost are two attributes and release time is the alternative in the framework of MAUT. Given that risk reduction and penalty cost control are both subjective, the single utility function is used to reveal management's own preference towards each attribute, i.e., risk and penalty cost. By allocating different importance weights for these two attributes, management can use the multi-attribute utility function to measure the attractiveness of the conjoint outcome of the risk and the penalty cost given a specified release time.

Another reason for the selection of MAUT is that it has strong theoretical foundations due to the use of the expected utility theory. The utility theory not only allows us to quantify management's preference towards each attribute with flexibility, but also takes management's risk structure into account, such as risk neutrality, risk aversion and risk proneness. Furthermore, MAUT provides a feasible approach for considering the continuous scale of the alternatives. Specifically, in our problem, the release time as the alternative should be considered in a continuous scale. Last but not least, when management has other requirements, i.e., the minimization of the total cost in the software development cycle (Sgarbossa and Pham, 2010), the control of the uncertainty in the total cost function (Yang et al., 2008), the optimized resource allocation (Ngo-The A and Ruhe, 2009), our decision model can be extended by introducing more attributes in the framework of MAUT. The proposed MAUT procedure for our decision problem is discussed in detail below.

6.3.1 Quantification of Attributes

The decision-maker should be identified before the application of our proposed MAUT procedure, and management here refers to the decision-maker(s) in our release time determination problem. In real applications, this decision maker is generally the quality manager of software products. When determining the release time, reducing the risk and controlling the penalty cost associated with it are both important for management.

On one hand, management is concerned about the risk that software cannot meet its reliability target due to parameter uncertainty. As shown in Figure 6.1, when the mean value of release time is used, there is 50% risk that software cannot meet the reliability requirement. Since the reliability requirement is generally set by customers, and it is usually specified in the sales/service contract, such risk can be too large to be acceptable. As a result, the risk, which is quantified in (6.8), should be minimized.

However, reducing this risk inevitably causes a delay of the release, and such delay can increase the cost during the testing phase. Provided that there is always limited budget for the testing process, controlling this delay penalty cost is also of great importance. Therefore, on the other hand, the penalty cost, which is quantified in (6.11), is also to be minimized from the management's point of view. It is worth noting here that the cost components $C_1(t)$ and $C_2(t)$ are considered for illustrative purpose. In practical applications, other cost components incurred can be added in a straightforward manner.

6.3.2 Elicitation of Single Utility Function for Each Attribute

After the quantification of each attribute, management's preference towards the performance of each attribute should be assessed. To represent this, the single utility function for each attribute is used. Suppose that the utility function for risk is to be determined, the worst and best values of risk are first selected as r_0^0 and r_0^1. In real applications, they represent the lowest risk requirement and the highest risk reduction expectation from management. For example, suppose that management can only accept a risk level below 5%, and the smaller the risk the better, until this risk can be eliminated. Hence, the lowest risk requirement is $r_0^0 = 5\%$ and the highest risk reduction expectation is $r_0^1 = 0$. At these boundary points, $u\left(r_0^0\right) = 0$ and $u\left(r_0^1\right) = 1$. The superscript of r_0^i is used to represent the corresponding utility value under the parameter and $i \in [0,1]$.

Subsequently, management is presented with some simple hypothetical gambles to determine the certainty equivalents for a few 50-50 lotteries (Keeney and Raiffa, 1976, von Winterfeldt and Edwards, 1986). For example, management is asked to chose a value for $r_0^{0.5}$, so that it is indifferent between accepting $r_0^{0.5}$ with certainty and having a 50-50 lottery, where there are 0.5 probabilities of getting r_0^0 and r_0^1 respectively. Similarly, $r_0^{0.75}$ can be determined with a 50-50 lottery which consists of $r_0^{0.5}$ and r_0^1. Also, $r_0^{0.25}$ can be obtained with a 50-50 lottery which includes r_0^0 and $r_0^{0.5}$. These five points are commonly used to elicit the single utility function for each attribute, which is generally represented by the linear or exponential function as follows (Keeney and Raiffa, 1976):

$$u(r_0) = \alpha + \beta \cdot r_0 \text{ or } u(r_0) = \alpha + \beta \exp(\gamma \cdot r_0) \ , \tag{6.13}$$

where α, β and γ are constants which ensure $u(r_0) \in [0,1]$. It should be noted that we also need to compare the certainty equivalents and the expected values of the 50-50 lotteries to determine which form in (6.13) should be selected. Specifically, if they are equal to each other, management is risk neutral and the linear form should be used. Otherwise, management is not risk neutral and the exponential form is generally adopted.

Similarly, the single utility function for penalty cost can be obtained following the procedure discussed above. First of all, C_p^0 and C_p^1 should be determined. For management, these values are of great importance because they represent their maximum penalty cost budget and the highest penalty cost control expectation. Then some points on the utility curve are assessed to obtain the single utility function for penalty cost.

6.3.3 Estimation of Scaling Constants

The following step is the estimation of the scaling constants w_1 and w_2, which represent the different importance weights allocated for risk and penalty cost respectively (von Winterfeldt and Edwards, 1986). There are two common methods to assess the scaling constants: certainty scaling and probabilistic scaling (von Winterfeldt and Edwards, 1986). Given that only two attributes are considered in our problem, and this is a small number, the probabilistic scaling technique is recommended.

When using the probabilistic scaling approach, management is asked to compare its preference between the two choices as shown in Figure 6.2. On the left hand side, there is a certain joint outcome $\left(r_0^1, C_p^0\right)$ comprising of risk at its best level and penalty cost at its worst level. On the right hand side, the lottery comprising of both attributes at their best levels with probability *p* and both attributes at their worst levels with probability 1-*p*.

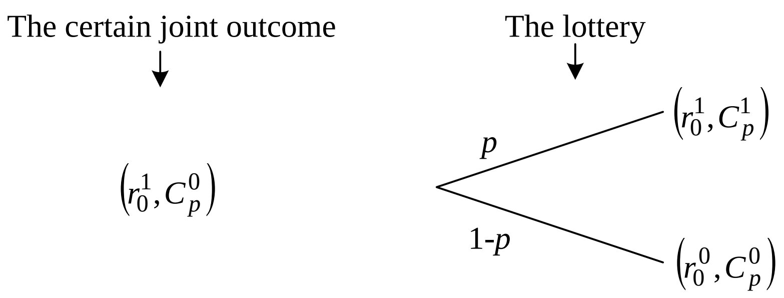

The certain joint outcome
The lottery
$\left(r_0^1, C_p^0\right)$
p
$1-p$
$\left(r_0^1, C_p^1\right)$
$\left(r_0^0, C_p^0\right)$

Figure 6.2 Two choices for the determination of the scaling constant w_1

Management is first asked to compare the certain outcome with the lottery having a 50-50 chance of occurring. If management prefers the certain outcome, the probability p is gradually increased until management is indifferent with these two choices. On the other hand, if management prefers the lottery, we decrease the probability p. At indifference, p is equal to the scaling constant w_1 for the risk attribute (von Winterfeldt and Edwards, 1986). Since the sum of the scaling constants must equal to one, w_2 can be obtained with ease.

6.3.4 Maximization of Multi-Attribute Utility Function

As discussed, the attractiveness of each attribute is measured by the single utility function based on management's own preference. After that, given different importance weights allocated for attributes, a one-dimensional multi-attribute utility function is constructed to reveal the attractiveness of the conjoint outcome of attributes given a specified alternative. The additive form of the multi-attribute utility function in our problem can be written as

$$U\left(r_0(t), C_p(t)\right) = w_1 u\left(r_0\right) + w_2 u\left(C_p\right) \tag{6.14}$$

where w_1 and w_2 are the scaling constants for attribute risk and penalty cost respectively and $u(r_0)$ and $u(C_p)$ are the single utility function for each attribute. By maximizing the multi-attribute utility function, the optimal risk-based release time is obtained as $T_R^* = \arg\max_t \{U(r_0(t), C_p(t))\}$.

It is worth noting here that the additive form of the multi-attribute utility function above is based on some independence assumptions and interested readers can refer to Keeney and Raiffa (1976) for more detailed theoretical discussions. In real applications, these assumptions are commonly accepted (Brito and Almeida, 2009; Ferreira et al., 2009). Moreover, it has been shown that even when these assumptions are violated, the additive multi-attribute utility function can provide fairly good approximations (Edwards, 1977; Farmer, 1987).

6.3.5 Summary of the Procedure

The procedure of our proposed MAUT approach in the decision problem is summarized in Figure 6.3. The first step of the implementation of the decision model is to quantify the attributes, i.e., the risk and the penalty cost. For the risk attribute, based on the standard statistical results, risk can be quantified by (6.8). For the attribute penalty cost, the generalized cost model is used and it is quantified by (6.11). The following step is the elicitation of single utility functions for both attributes. After this, the scaling constants for each attribute are estimated by comparing the two choices as shown in Figure 6.2. Finally, based on the single utility functions and the scaling constants, the multi-attribute utility function is obtained as shown in (6.14).

The optimal risk-based release time, which is the best option of release time in terms of risk and penalty cost, is determined by maximizing it.

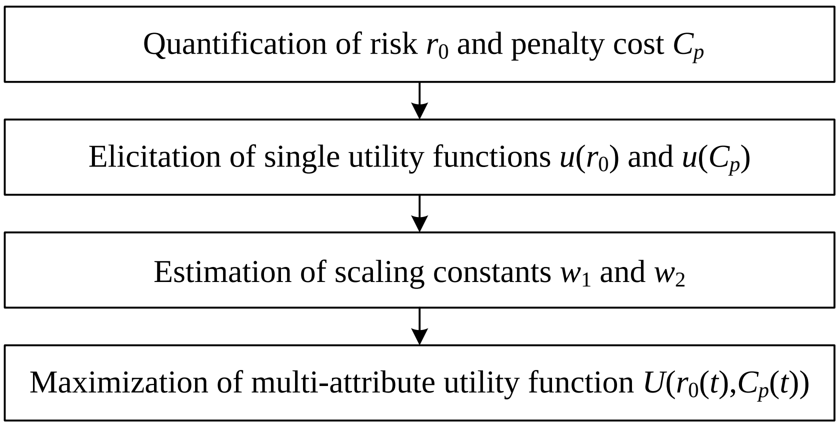

Quantification of risk r_0 and penalty cost C_p
Elicitation of single utility functions $u(r_0)$ and $u(C_p)$
Estimation of scaling constants w_1 and w_2
Maximization of multi-attribute utility function $U(r_0(t),C_p(t))$

Figure 6.3 The structure of the decision model for the determination of optimal risk-based release time

6.4 An Illustrative Example

In this section, a decision model application example is provided for illustrative purpose. By considering the risk and the penalty cost simultaneously, optimal risk-based release time is determined by incorporating management’s own preference into the decision process. In addition, sensitivity analysis is introduced to help management check the robustness of the final decision.

6.4.1 The Data Set

In this example, the data set used in Pham and Zhang (1999) is adopted. The reason for this selection is that both failure data and cost parameters are provided in it. In particular, based on the failure data, the quantification of risk can be done; for the cost parameters, they can be used to obtain the general penalty cost function and we have c_1=700, $\kappa = 0.95$, c_2=60 and $\mu_y = 0.1$.

It should be noted here that the estimates of these cost parameters are usually determined based on previous experiences or expert opinions. Therefore, physical meanings of these parameters are of great importance and they are illustrated here again as follows: c_1 is the software test cost per unit time, κ is the discount rate of the testing cost due to the learning effect, c_2 is the cost of removing an error per unit time during the testing phase and μ_y is the expected time of removing an error during this period. Different software projects usually generate different estimates of these parameters. However, the physical meanings of these parameters can ensure that they are estimated in a consistent way.

6.4.2 The Determination of Optimal Risk-Based Release Time

Following the procedure discussed in Section 6.3, the determination of optimal risk-based release time is shown in a step-by-step manner.

Step 1: Quantification of risk and penalty cost

The Goel-Okumoto (GO) model (Goel and Okumoto, 1979) is adopted in Pham and Zhang (1999) to analyze the failure data for reliability assessment. In this study, we adopt this model as well. The mean value function and the failure intensity function of the GO model are given by

$$m(t) = a(1 - e^{-bt}) \text{ and } \lambda(t) = abe^{-bt} \tag{6.15}$$

where a denotes the number of expected faults in the software and b represents the fault detection rate. Furthermore, the reliability of the software system is obtained as

$$R(x \mid t) = \exp\left[-\lambda(t)x\right] \tag{6.16}$$

and $R(x \mid t)$ represents the conditional software reliability, which is defined as the probability that the software will not fail given a specified time interval $(t, t + x]$ in the operational phase (Yang and Xie, 2000). We set x equal to 1 without loss of generality. Then the release time based on the reliability target R_0 is

$$T = \frac{1}{b}\ln\left[\frac{ab}{\ln(1/R_0)}\right] \tag{6.17}$$

Suppose that customer has indicated a reliability requirement of R_0=0.95. Based on the maximum likelihood estimates as $\hat{a} = 142.32$ and $\hat{b} = 0.1246$, the mean value of the release time is $\hat{T} = 46.91$. Moreover, based on the standard statistical analysis as shown in Section 6.1, we have $Var(\hat{a}) = 154.85$, $Var(\hat{b}) = 2.17 \times 10^{-4}$, $Cov(\hat{a}, \hat{b}) = -0.0358$. Hence, the variance of the release time

$$
\begin{aligned}
Var(\hat{T}) &= \left(\frac{\partial T}{\partial a}\right)^2_{\substack{a=\hat{a}\\ b=\hat{b}}} Var(\hat{a}) + \left(\frac{\partial T}{\partial b}\right)^2_{\substack{a=\hat{a}\\ b=\hat{b}}} Var(\hat{b}) + 2\left(\frac{\partial T}{\partial a}\right)\left(\frac{\partial T}{\partial b}\right)_{\substack{a=\hat{a}\\ b=\hat{b}}} Cov(\hat{a},\hat{b}) \\
&= \left(\frac{1}{\hat{a}\hat{b}}\right)^2 Var(\hat{a}) + \frac{1}{\hat{b}^4}\left\{1+\ln\left[\frac{\ln(1/R_0)}{\hat{a}\hat{b}}\right]\right\}^2 Var(\hat{b}) \qquad (6.18) \\
&+ \frac{2}{\hat{a}\hat{b}^3}\left\{1+\ln\left[\frac{\ln(1/R_0)}{\hat{a}\hat{b}}\right]\right\} Cov(\hat{a},\hat{b})
\end{aligned}
$$

is obtained as $Var(\hat{T}) = 22.35$. Accordingly, the attribute risk can be quantified by substituting these estimated parameters into (6.8).

For the quantification of the penalty cost function, it is relatively simple by substituting estimated values of cost parameters into (6.11), and our decision space for the release time is $t \in [46.91, +\infty)$.

Step 2: Elicitation of single utility functions

The following step is to assess management's preference towards the performance of each attribute, i.e., the risk and the penalty cost. Interviews with management are needed to elicit reasonable single utility functions.

Suppose that management scenarios are as follows:

(1) Management has verified that it is risk neutral towards both attributes.

(2) Management indicate that it can only accept up to a risk level of 5%, and the smaller the risk the better, until this risk can be eliminated.

(3) Management has an additional penalty cost budget of \$15000 and it is completely unsatisfied when all the money is spent; its satisfaction increases when the money

spent decreases, and the highest satisfaction level is achieved when no money is spent.

Based on the management scenarios above, corresponding explanations on the determination of single utility functions are shown as follows:

(1) Since management is risk neutral towards both attributes, the linear form of the single utility function should be used.

(2) The lowest risk requirement is $r_0^0 = 5\%$ and the highest risk reduction expectation is $r_0^1 = 0$. The single utility function for risk is obtained as $u(r_0) = 1 - 20r_0$.

(3) The maximum penalty cost budget is $C_p^0 = 15000$ and the highest penalty cost control expectation is $C_p^1 = 0$. The single utility function for penalty cost is determined as $u(C_p) = 1 - C_p/15000$.

Step 3: Estimation of scaling constants

In this stage, the scaling constant w_1 is estimated first by comparing the two choices in Figure 6.2. Suppose management claims that it is indifferent between these two choices when p is equal to 0.5. Then, w_1=0.5. Since the sum of scaling constants is equal to one, w_2 is equal to 0.5 as well.

Step 4: Maximization of multi-attribute utility function

Based on the estimated single utility functions and scaling constants, the multi-attribute utility function can be obtained by (6.14). Figure 6.4 shows this multi-attribute utility function as a function of the release time. This multi-attribute utility function is maximized when $T_R^* = 60.15$ and the corresponding risk and penalty cost at this time are $r_0\left(T_R^*\right) = 0.28\%$ and $C_p\left(T_R^*\right) = 7216$ respectively. As a result, software should be released at the optimal risk-based release time $T_R^* = 60.15$ to appropriately compromise between reducing the risk and controlling the penalty cost.

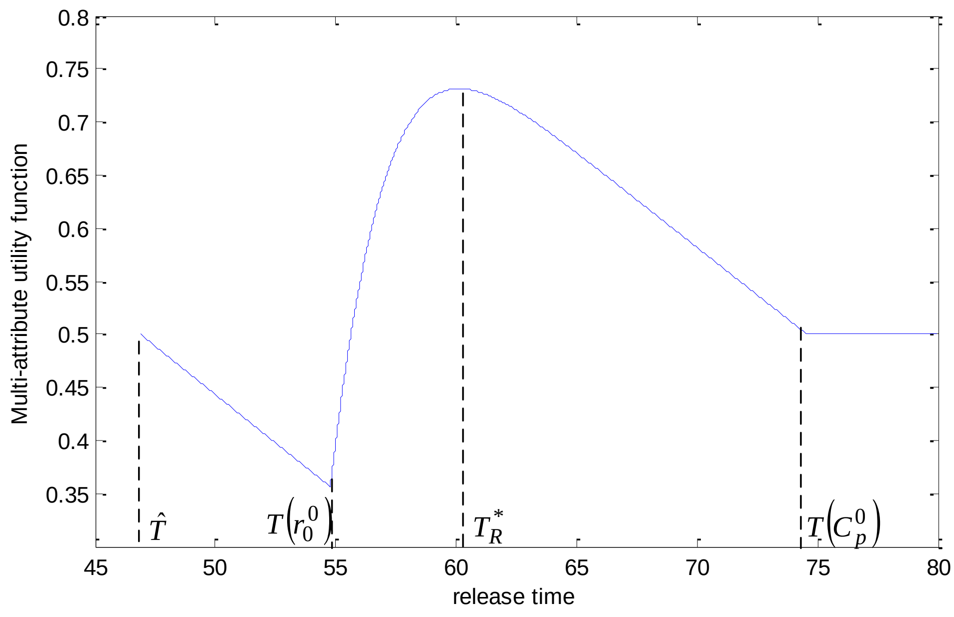


Figure 6.4 Multi-attribute utility function given different release times

6.4.3 Illustration of the Proposed Decision Model

In Figure 6.4, we denote $\hat{T} = 46.91$ as the mean value of release time without consideration of parameter uncertainty. If we release the software at this time, no penalty cost is incurred and the highest penalty cost control expectation $C_p^1 = 0$ is satisfied. However, at this release time, the 50% risk is too high to be acceptable for management because the lowest risk requirement $r_0^0 = 5\%$ is not satisfied. At this point, management has to make a compromise between reducing the risk and controlling the penalty cost.

With the consideration of this, the software testing is expected to increase. We denote $T\left(r_0^0\right) = 54.77$ and $T\left(C_p^0\right) = 74.59$ as the release times when the lowest risk requirement $r_0^0 = 5\%$ and the maximum penalty cost budget $C_p^0 = 15000$ are satisfied respectively. These time points are of great importance since both attributes will contribute to the multi-attribute utility function during this time period. It is found that at the optimal risk-based release time $T_R^* = 60.15$, the multi-attribute function is maximized. In other words, when both risk and penalty cost are considered, a compromise can be made to optimize them simultaneously, and corresponding risk and penalty cost are $r_0\left(T_R^*\right) = 0.28\%$ and $C_p\left(T_R^*\right) = 7216$ respectively.

Finally, it should be noted that during the time periods $\left[\hat{T}, T\left(r_0^0\right)\right]$ and $\left[T\left(C_p^0\right), +\infty\right)$, the multi-attribute utility function is dominated by only one attribute. More specifically, for the first period, since the lowest risk requirement $r_0^0 = 5\%$ has not been satisfied, the penalty cost is the only attribute contributing to the multi-attribute utility function. Given that the penalty cost is increasing over time and management's satisfaction level is decreasing with it, the multi-attribute utility function is decreasing during this time period. While for the second time period, the multi-attribute utility function is dominated by the risk attribute and it is equal to $0.5u(r_0)$. Figure 6.4 shows that the multi-attribute utility function remains at 0.5 level when release time is greater than

$T\left(C_p^0\right)$. This implies that the available penalty cost budget $C_p^0 = 15000$ is sufficient for management to reduce the risk to the best level $r_0^1 = 0$.

6.4.4 Sensitivity Analysis

Optimal risk-based release time can be determined by maximizing the multi-attribute utility function. However, since most parameters in the MAUT are obtained based on the subjective assessments of management, the optimal risk-based release time obtained may not be accurate. In practice, management has to know how robust the optimal decision is, and sensitivity analysis is needed. More specifically, sensitivity analysis can help to investigate the relative change of the optimal solution when a specific parameter changes, i.e., the change of cost parameters, scaling constants, etc. The results from sensitivity analysis reveal the stability of the optimal solution.

Sensitivity analysis is generally done by changing one parameter and setting the other parameters at their fixed values (Xie and Hong, 1998; Li et al., 2010). When parameter *x* is investigated to see how much the optimal risk-based release time T_R^* changes, T_R^* is in fact a function of *x* as other parameters are fixed using their estimated values. The sensitivity of the optimal decision to this parameter can be quantified by $S_{q,x}$, defined as the relative change of the optimal risk-based release time when *x* is changed by 100*q*% (Xie and Hong, 1998; Li et al., 2010).

$$S_{q,x} = \frac{T_R^*(x+qx) - T_R^*(x)}{T_R^*(x)} \tag{6.19}$$

A large value of $S_{q,x}$ indicates that parameter x has significant influence on the determination of T_R^*, and T_R^* is regarded as sensitive to the change of x. Normally, management should pay special attention to the significant parameters as the optimal decision T_R^* is heavily dependent on the accurate estimates of them (Xie and Hong, 1998; Li et al., 2010).

From a practical point of view, it may not be necessary to conduct sensitivity analysis for all the parameters in this optimal release time problem. For instance, parameters c_2 and μ_y are expected to be insignificant. The reason is that the expected cost to remove errors from time t to $\hat{T}$ is negligible in (6.11). More specifically, given a high reliability requirement such as R_0=0.95 in our application example, there will be few faults detected from $\hat{T}$ to t. Additionally, as c_1=700, c_2=60 and $\mu_y = 0.1$; compared with the estimated value of c_1, the product of c_2 and μ_y is too small to have any impact on the penalty cost function in (6.11). Another example is the determination of r_0^1 and C_p^1, which represent the highest risk reduction expectation and highest penalty cost control expectation respectively. Since management always prefer less risk and less cost, setting them to zero can properly describe the best cases for risk control and penalty cost control respectively.

In contrast, parameters c_1 and κ are much more important since they dominate the change of the penalty cost over time. Similarly, r_0^0 and C_p^0 are of great importance as shown in Figure 6.4, where $T\left(r_0^0\right)$ and $T\left(C_p^0\right)$ are change points of multi-attribute utility function. Furthermore, scaling constants w_1 and w_2 are also important since they denote the different importance weights allocated for each attribute, which directly affect the final solution on T_R^*. However, since the sum of these two weights is equal to one, investigating one factor is sufficient. Results of sensitivity analysis with regard to these parameters are summarized in Table 6.1. Specially, since parameter κ represents the learning effect of the testing team which is not greater than one, the value of $S_{q,\kappa}$ when $\kappa = 1$ is used for the positive change of κ.

Table 6.1 Sensitivity analysis results given different parameters

q	-30%	-20%	-10%	10%	20%	30%
S_{q,w_1}	-0.0186	-0.0120	-0.0058	0.0057	0.0114	0.0171
S_{q,C_p^0}	-0.0105	-0.0065	-0.0030	0.0027	0.0052	0.0074
S_{q,r_0^0}	0.0100	0.0063	0.0030	-0.0028	-0.0053	-0.0077
S_{q,c_1}	0.0100	0.0063	0.0030	-0.0028	-0.0053	-0.0077
$S_{q,\kappa}$	0.0404	0.0272	0.0138		-0.0075	

It can be seen that these parameters do not significantly influence the final solution on T_R^* since all the absolute values of $S_{q,x}'s$ are below 5%. In other words, the optimal risk-based release time obtained is robust to changes in the parameters. For example, when parameter κ decreases by 30 percent, the relative change of T_R^* is only about 4 percent. Moreover, results in Table 6.1 indicate that T_R^* is positively correlated with w_1 and C_p^0, and negatively correlated with r_0^0, c_1 and κ. Physical meanings of these parameters can actually explain these results. For instance, when w_1 increases, it means that more importance is allocated for the control of risk. As a result, T_R^*

increases as well. Last but not least, it is interesting that parameter r_0^0 and c_1 appears to have the same effect on T_R^*. This result will be explained in the following section, where a simplification of the decision model is discussed.

6.5 A Simplification of the Decision Model

The decision model proposed can be simplified to provide analytical tractability with some additional non-restrictive assumptions. These assumptions are summarized as follows:

(1) Management is risk neutral towards each single attribute, i.e., the risk and the penalty cost;

(2) Management set its highest risk reduction expectation as $r_0^1 = 0$ and its highest penalty cost control expectation as $C_p^1 = 0$;

(3) The penalty cost is dominated by the general testing cost $C_1(t)$ and the learning effect of the testing team is negligible such that $\kappa = 1$.

From a practical point of view, these assumptions may not appear too far-fetched or restrictive. The first assumption is a widely adopted assumption in practice, especially when the single utility function is estimated empirically (Scholz and Tietje, 2002). For the second assumption, since management always prefer less risk and less cost, setting them to zero can properly describe the best cases for risk control expectation and penalty cost control expectation. As for the third assumption, the preceding illustrative example has revealed that the penalty cost is indeed dominated by the general testing cost and the optimal risk-based release time is not sensitive to the change of the learning effect factor.

Based on these additional assumptions, if $T\left(r_0^0\right) < T\left(C_p^0\right)$, the multi-attribute utility function when $t \in \left[T\left(r_0^0\right), T\left(C_p^0\right)\right]$ simplifies to

$$U(t) = w_1 \frac{r_0^0 - r_0(t)}{r_0^0} + w_2 \frac{C_p^0 - C_p(t)}{C_p^0} \tag{6.20}$$

where $C_p(t) = c_1\left(t - \hat{T}\right)$.

Theorem: When $t \in \left[T\left(r_0^0\right), T\left(C_p^0\right)\right]$ and $T\left(r_0^0\right) < T\left(C_p^0\right)$, define $k = r_0^0 \frac{w_2}{w_1} \frac{c_1}{C_p^0} \sqrt{Var\left(\hat{T}\right)}$.

If $k \le \varphi\left(T\left(r_0^0\right)\right)$, the multi-utility function for the simplified model is maximized at $t^* = \hat{T} + \sqrt{-Var\left(\hat{T}\right)\ln\left(2\pi k^2\right)}$; if $k > \varphi\left(T\left(r_0^0\right)\right)$, $t^* = T\left(r_0^0\right)$.

Proof: Substitute (6.15) into (6.14) and take the first and second derivative of multi-attribute utility function with respect to *t*, we have

$$\frac{dU(t)}{dt} = \frac{w_1}{r_0^0 \sqrt{Var(\hat{T})}} \varphi\left[\frac{t - \hat{T}}{\sqrt{Var(\hat{T})}}\right] - w_2 \frac{c_1}{C_p^0} \tag{6.21}$$

and

$$\frac{d^2U(t)}{dt^2} = -\frac{w_1\left(t-\hat{T}\right)}{r_0^0 Var(\hat{T})}\varphi\left[\frac{t-\hat{T}}{\sqrt{Var(\hat{T})}}\right] \tag{6.22}$$

Since the risk-based release time is always greater than the mean value $\hat{T}$, $d^2U(t)/dt^2$ is not greater than zero for all *t*. If $k \le \varphi\left(T\left(r_0^0\right)\right)$, there is a feasible solution

$$t^* = \hat{T} + \sqrt{-Var\left(\hat{T}\right)\ln\left(2\pi k^2\right)}, \tag{6.23}$$

under which $dU(t)/dt = 0$ and the multi-attribute utility function is maximized. Otherwise, if $k > \varphi\left(T\left(r_0^0\right)\right)$, $dU(t)/dt < 0$ which indicates that the multi-attribute utility function is a decreasing function, and it is maximized at $t^* = T\left(r_0^0\right)$. □

Subsequently, for $t \in \left[\hat{T}, T\left(r_0^0\right)\right)$, it has been discussed that during this time, the multi-attribute utility function is only determined by the single utility function for the penalty cost and it is decreasing over time. The maximum value of it is equal to *w*2 and it is achieved at $\hat{T}$. Similarly, for $t \in \left(T\left(C_p^0\right), T\left(r_0^1\right)\right]$, the maximum value of the multi-attribute utility function is *w*1 under $T\left(r_0^1\right)$.

Accordingly, under the condition that $T\left(r_0^0\right) < T\left(C_p^0\right)$, management can determine optimal risk-based release time T_{SR}^* with the simplified decision model easily. It is known that at release times $T\left(r_0^1\right)$, $\hat{T}$ and t^*, corresponding values of the multi-attribute function are w_1, w_2 and $U\left(t^*\right)$ respectively. T_{SR}^* is then selected among these three release times, under which the corresponding multi-attribute utility value is the largest. Mathematically, by defining $\vec{T} = \left\lfloor T\left(r_0^1\right), \hat{T}, t^* \right\rfloor$, we have $T_{SR}^* = \arg\max_{t \in \vec{T}} U(t)$.

Previous discussions are based on the condition $T\left(r_0^0\right) < T\left(C_p^0\right)$ and it means that the maximum penalty cost budget is sufficient enough to achieve the minimum risk requirement. While under the condition that $T\left(r_0^0\right) \geq T\left(C_p^0\right)$, during the time period $t \in \left\lfloor T\left(C_p^0\right), T\left(r_0^0\right) \right\rfloor$, the value of the multi-attribute function is equal to zero because the maximum cost budget is exceeded and the lowest risk requirement is not achieved. Accordingly, only one attribute can be optimized in this case. Management needs to compare values w_1 and w_2, which represent the importance weights allocated for risk and penalty cost respectively. If $w_1 > w_2$, $T_{SR}^* = T\left(r_0^1\right)$; otherwise, $T_{SR}^* = \hat{T}$.

The structure of the simplified decision model is essentially the same as that of the general decision model as shown in Figure 6.3. However, some changes are made in the first two steps. For the first step, the penalty cost function is simplified based on the assumption (3). For the second step, since the assumptions (1) and (2) are adopted, the linear form is used to represent the single utility function. Due to these changes, the complexity of the decision process is greatly reduced, and the determination of optimal risk-based release time with the simplified decision model is shown in Figure 6.5.

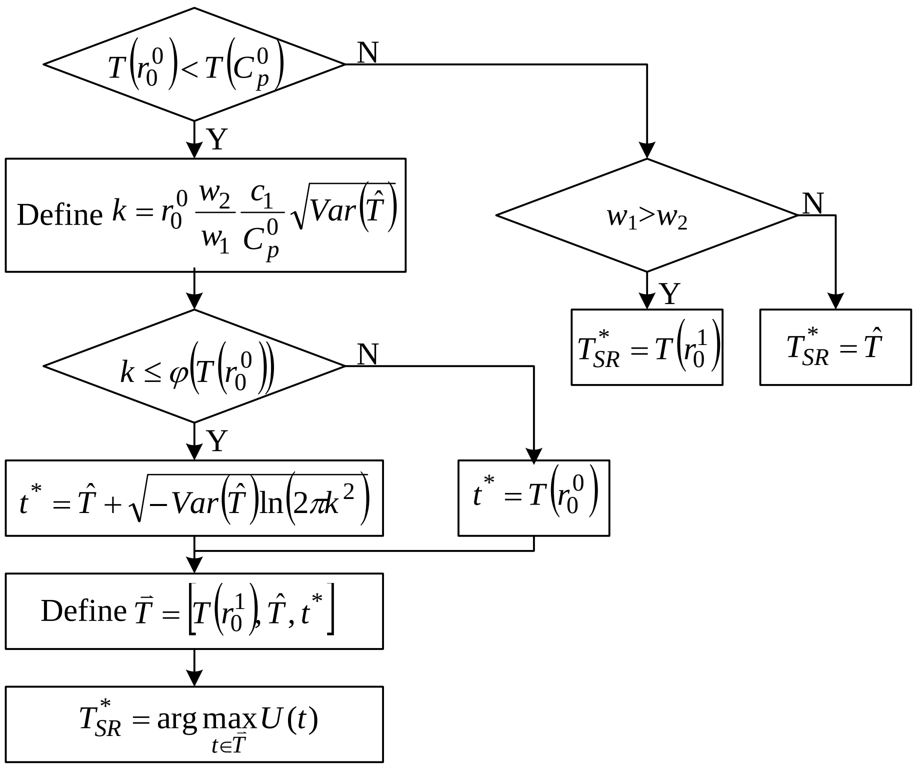

$T(r_0^0) < T(C_p^0)$
N
Y
Define $k = r_0^0 \frac{w_2}{w_1} \frac{c_1}{C_p^0} \sqrt{Var(\hat{T})}$
$w_1 > w_2$
N
Y
$T_{SR}^* = T(r_0^1)$
$T_{SR}^* = \hat{T}$
$k \le \varphi(T(r_0^0))$
N
Y
$t^* = \hat{T} + \sqrt{-Var(\hat{T}) \ln(2\pi k^2)}$
$t^* = T(r_0^0)$
Define $\vec{T} = [T(r_0^1), \hat{T}, t^*]$
$T_{SR}^* = \arg\max_{t \in \vec{T}} U(t)$

Figure 6.5 Determination of the optimal risk-based release time under the simplified decision model

Illustrating with the example in Section 6.4, we see that the first condition $T(r_0^0) < T(C_p^0)$ is just satisfied. Moving to the next step, since k=0.0112 and $\varphi[T(r_0^0)] = 0.1031$, the condition $k \le \varphi[T(r_0^0)]$ is satisfied. Therefore $t^* = \hat{T} + \sqrt{-Var(\hat{T}) \ln(2\pi k^2)} = 59.70$ which gives $U(t^*) = 0.6643$. As a result of $U(t^*) > w_1 = w_2 = 0.5$, the optimal risk-based release time based on the simplified decision model is $T_{SR}^* = t^* = 59.70$. Compared with $T_R^* = 60.15$ obtained in Section 5.4.2, there is only -0.75% relative difference. This implies that the simplified decision model (under the additional assumptions) can provide a fairly good approximation.

Based on this approximation, the sensitivity analysis results given parameter r_0^0 and c_1 can be explained as well. Since $T_{SR}^* = t^* = \hat{T} + \sqrt{-Var(\hat{T})\ln(2\pi k^2)}$ and $k = r_0^0 \frac{w_2}{w_1}\frac{c_1}{C_p^0}\sqrt{Var(\hat{T})}$, we have $T_{SR}^*(r_0^0 + qr_0^0) = T_{SR}^*(c_1 + qc_1)$ and $S_{q,r_0^0} = S_{q,c_1}$. Accordingly, parameters r_0^0 and c_1 appear to have the same effect on the final solution of T_R^* as shown in Table 6.1.

6.6 Threats to Validity

Based on the standard statistical analysis (Nelson, 1982), there is 50% chance that the software will not meet its reliability requirement when the mean value $\hat{T}$ is used. However, it should be noted that the standard statistical analysis is for approximation. It is still an open question whether the risk is really as high as 50%. To investigate this problem, an empirical case study is conducted by the Monte Carlo simulation using MATLAB software.

In particular, the GO model is adopted, where the preset parameters are given by $a = 100$ and $b = 0.1$. Therefore, suppose that the reliability requirement is $R_0 = 0.95$, the real value of optimal release time is $T_{real} = 52.73$. According to the general procedures discussed in Lyu (1996), 10000 failure data sets are generated, and each failure data set is composed of ninety time to failures data. Since each failure data set can produce an estimate of the optimal release time denoted by $\hat{T}$, risk that software cannot meet the reliability requirement can be easily estimated by comparing these $\hat{T}$

values with T_{real}, and such risk is estimated as $\hat{r}_0 = 60.21\%$. Although this result is different from the estimated risk based on the standard statistical analysis, it severs as another piece of evidence that the risk due to parameter uncertainty cannot be neglected.

In addition, previous discussions are based on the closed form of the mean value function given by (6.17). It is possible that there is no closed form of the mean value function if a different model instead of GO model is used for the analysis, e.g., the S-shaped model (Yamada et al., 1983). When there is no closed form of optimal release time, the variance of it cannot be computed analytically as in (6.5) and (6.18). In this case, the Monte Carlo simulation approach could be a good alternative as it has been widely and successfully used in the uncertainty analysis of many complex systems (Helton and Davis, 2003). In general, such analysis can be regarded as the study of functions of the form

$$\mathbf{y} = f(\mathbf{x}) \tag{6.24}$$

where $\mathbf{x} = [x_1, x_2, ...x_m]$ is a vector of analysis inputs and $\mathbf{y} = [y_1, y_2, ...]$ is a vector of outputs. To evaluate the uncertainty of the elements of **y**, uncertainty of **x** is supposed to be known in advance and it is generally characterized by a sequence of probability distributions denoted by $D_1, D_2, ..., D_m$ for each element in **x** respectively. According to the distributions of **x** and other associated restrictions, samples of inputs are generated and the corresponding values of outputs are received. Then, cumulative distribution functions (CDFs) for **y** can be estimated and uncertainty in **y** is analyzed based on these CDFs.

For our problem, only optimal release time is the output of interest and it can be written as

$$T = f(\boldsymbol{\theta}) \tag{6.25}$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_m]$ is the vector of input parameters. Based on the discussion of Section 6.1, input parameters can be regarded as normally distributed random variables and their mean values and variances can also be estimated. Further, with the use of asymptotic covariance matrix given by (6.3), correlated Gaussian random numbers can be generated following the standard procedures discussed in Johnson (1987). Let N denote the base sample size and therefore there will be N values of the optimal release time given by

$$T_i = f(\theta_{1i}, \theta_{2i}, \cdots, \theta_{mi}) \quad i = 1,2,\ldots,N \tag{6.26}$$

Hence, the risk can be determined according to the definition given by (6.8) and the estimated CDF of T. More specifically, the CDF of T given a determined value of t can be estimated as

$$r_0(t) = P\{T > t\} = \sum_{i=1}^{N} \delta_T(T_i) \frac{1}{N} \tag{6.27}$$

where

$$\delta_T(T_i) = \begin{cases} 0 & T_i < t \\ 1 & T_i \geq t \end{cases} \tag{6.28}$$

Another possible limitation for the quantification of risk is that the normal distribution is used to quantify the parameter uncertainty. Although this kind of approximation technique is widely adopted in reliability engineering, it may not be accurate. In this case, incorporating experts' opinion and past experience could be a choice. For example, experts could probably know the distributions of some model parameters based on their past experience on similar software projects. Based on this kind of information, parameter uncertainty can be quantified effectively by combining the Maximum-Entropy Principle (MEP) into the Bayesian approach as discussed in Dai et al. (2007).

Besides the consideration of risk, the penalty cost associated with it is incorporated into our decision problem. This is because the risk cannot be overlooked due to the limited cost budget of the project (Nan and Harter, 2009). Management needs to strike a balance between reducing the risk and controlling the penalty cost associated with the risk. In other words, given a reliability requirement, we introduce two new important dimensions for the determination of optimal release time: the risk that software cannot meet the reliability requirement due to parameter uncertainty and the penalty cost associated with such risk. However, it should be noted that the formulation here may not be enough for release time determination. In reality, management can also have other requirements, which may include the minimization of the total cost in the software development cycle (Sgarbossa and Pham, 2010), the control of the uncertainty in the total cost function (Yang et al., 2008), and the optimized resource allocation (Ngo-The and Ruhe, 2009), etc. When these requirements are considered, our decision model should be extended by introducing more attributes in the framework of MAUT.

Last but not least, although the proposed decision model can better describe the management's perspective, it requires the model user to have the knowledge on how to apply MAUT in the decision problem properly. For large and experienced companies, this can be done in some training programs. While for the other companies, which may only require some empirical results, they can probably choose the simplified decision model.

6.7 Conclusion

The software release problem is of great importance in the software development cycle. In this chapter, when to release software given a reliability constraint is discussed in detail. In particular, we highlight the risk in the reliability estimate due to parameter uncertainty. However, reducing such risk inevitably increases the testing costs. Thus, from management's point of view, a compromise should be made between reducing the risk and controlling the delay penalty cost associated with it. Due to this consideration, a decision model based on MAUT is developed for the determination of optimal risk-based release time. The proposed model provides management with a boarder view of the release time determination problem. It not only allows management to optimize two conflicting criteria simultaneously, but also incorporates management's own preference into the decision process.

The decision model proposed in this chapter is also general in terms of applicability since different software reliability models and cost models can be used in the testing process. Furthermore, the proposed decision model can be simplified under some non-restrictive assumptions. The simplified decision model not only provides analytical tractability, but also greatly reduces the complexity of the decision process. Since the MAUT approach is sometimes criticized for its complex decision process, the simplified decision model can probably provide a good alternative, especially when some empirical results are needed.

Chapter 7 Multi-Objective Optimization Approaches to Software Release Time Determination

7.1 Basic Problem Description

For optimal release time determination problem, it is generally formulated in one of the following ways: cost minimization, cost minimization given a reliability target, and reliability maximization under a cost budget. Obviously, all of these three optimization models formulate the optimal release time problem as a single-objective optimization problem. Although these formulations are simple to use, they cannot describe management's preference accurately. In reality, it seems to be more reasonable to describe the management's attitude like this: maximizing reliability and minimizing cost are expected to be achieved simultaneously. Therefore, in this chapter, the decision problem is formulated as a multi-objective optimization problem, and different multi-objective optimization approaches are investigated.

The remainder of this chapter is organized as follows. In Section 7.2, multi-objective optimization model is formulated for software release time determination problem. In Section 7.3, different multi-objective optimization approaches, including the trade-off analysis, multi-attribute utility theory (MAUT), and physical programming approach, are introduced. In Section 7.4, two numerical examples are provided for illustrative purpose. In Section 7.5, applicability and limitations of these multi-objective optimization approaches are studied. Finally, concluding remarks are given in Section 7.6.

7.2 Model Formulation for Release Time Determination

In traditional formulations, it is difficult to make a priori selection of constraint values, i.e., the reliability target R_0 and the cost budget C_0. These constraint values will be modified frequently to obtain a satisfied solution of optimal release time, which is time-consuming and error-prone. Furthermore, optimal release time solutions under traditional formulations can be highly sensitive to the constraint values. We take the cost minimization given a reliability target R_0 as an example for illustration. In Figure 7.1, we denote t_0^* as the time at which the expected testing cost $E[C(t)]$ is minimized, and its corresponding reliability value is R_0^*. It can be easily seen from the figure that once the constraint value R_0 is greater than R_0^*, the optimal release time is completely determined by the constraint condition, and it is equal to t_0.

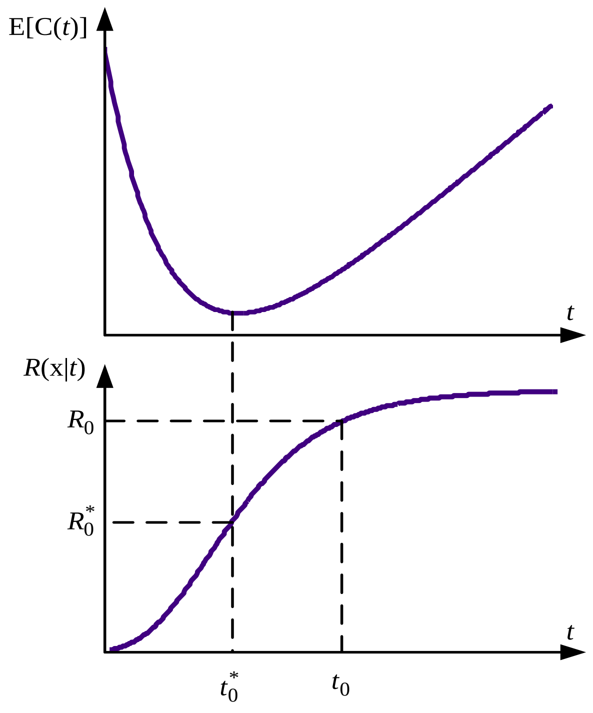


Figure 7.1 Relationship between $E[C(t)]$ and $R(x|t)$

It has been shown that the single-objective optimization models have many disadvantages when they are used to solve the optimal software release time problem. In reality, reliability and cost should be optimized simultaneously. Therefore, we formulate the release time determination problem as follows:

Formulation 1

$$\begin{cases} \text{Maximize} & R(x|t) \\ \text{Minimize} & E[C(t)] \\ \text{Subject to} & t > 0, x > 0 \end{cases}$$

In this formulation, t is the release time of software, and $R(x|t)$ represents the conditional software reliability which is defined as the probability that the software will not fail given a specified time interval $(t, t+x]$. $E[C(t)]$ is the expected cost at time *t*.

In the optimal software release problem, the evaluation of software reliability and expected cost is of great importance. Software reliability is generally measured based on a specific software reliability model (Musa et al., 1987; Xie, 1991; Pham, 2000). This model is selected based on the recorded failure data and experts' prior knowledge. Among software reliability models, non-homogeneous Poisson process (NHPP) models form a major part of it. Suppose that the mean value function of the NHPP is denoted by $m(t)$, the testing reliability of software is measured by

$$R(x|t) = e^{-[m(t+x)-m(t)]}. \tag{7.1}$$

The testing reliability concept is under the scenario that software will be still in the testing phase in the time interval $(t, t+x]$. However, from customers' point of view, software will not be tested after its release. The operational reliability of software is a more meaningful and appropriate reliability measurement in the context of release time determination (Yang and Xie, 2000). Therefore, in this study, we adopt the operational reliability concept for software reliability measurement, and the reliability of software is measured by

$$R(x|t) = e^{-[\lambda(t)x]}. \tag{7.2}$$

Besides the measurement of reliability, the expected cost is another major concern in the software development. In the literature, different kinds of cost models are developed. Among these cost models, the cost model proposed by Pham and Zhang (1999) is a general one, and most cost models are obtained based on the simplification of it (Yang et al., 2008; Sgarbossa and Pham, 2010). In particular, the general cost model is given by

$$E[C(t)] = c_0 + c_1 t^{\kappa} + c_2 \mu_y m(t) + c_3 \mu_w [m(t+t_w) - m(t)] + c_4 [1 - R(x|t)], \qquad (7.3)$$

where c_0 is the set-up cost for software testing, c_1 is the cost of testing per unit testing time, κ is the discount rate of testing cost over time ($0 < \kappa \le 1$), c_2 and c_3 are the cost of removing a fault per unit time in the testing phase and warranty phase respectively, μ_y and μ_w are expected time to remove a fault during the testing phase and warranty phase respectively, t_w is the warranty period, and c_4 is the cost due to software failure. Since removing a fault in the warranty phase is more expensive than that in the testing phase, c_3 is always greater than c_2. In addition, the parameter κ tries to capture the learning effect of the testing team.

It should be noted that management may also have other objectives to be optimized in the release time determination problem. For example, Xie et al. (2010) introduced the risk that software cannot meet its reliability requirement due to parameter uncertainty, and this risk is another important dimension that should be incorporated in the release time determination. Based on their study, when the mean value of release time $\hat{T}$ is used, there is as high as 50% risk that software reliability target cannot be met. Since such risk could be too high to be acceptable for management, it is expected to be reduced. Therefore, in this chapter, we will also study the multi-objective optimization problem when such risk is incorporated. Mathematically, this problem is formulated as

Formulation 2

$$\begin{cases} \text{Maximize} & R(x|t) \\ \text{Minimize} & E[C(t)] \\ \text{Minimize} & r(t) \\ \text{Subject to} & t > 0, x > 0 \end{cases}$$

where $r(t)$ represents the risk that software cannot meet its reliability target R_0 when software is released, and it is quantified as

$$r(t) = 1 - \int_{-\infty}^{\frac{t-\hat{T}}{\sqrt{Var(\hat{T})}}} \varphi(x)dx . \tag{7.4}$$

In equation (7.4), $\hat{T}$ is the estimated release time given the reliability target R_0, $Var(\hat{T})$ is the variance of $\hat{T}$, and $\varphi(x)$ is the probability density function of standard normal distribution. More detailed discussions on the calculation of the risk have been shown in Chapter 6.

In summary, in order to optimize various objectives simultaneously, release time determination is formulated as multi-objective optimization problems. To solve these two multi-objective optimization problems as shown in Formulation 1 and

Formulation 2, different multi-objective optimization approaches are adopted, which will be shown in the following section.

7.3 Multi-Objective Optimization Approaches

As discussed, for release time determination, it is an essentially multi-objective optimization problem. In this section, three widely used multi-objective optimization approaches are introduced for our multi-objective optimization problems, i.e., the trade-off analysis, MAUT, and physical programming method.

7.3.1 The Trade-Off Analysis

The objective of trade-off analysis is to identify the non-dominated solutions to the multi-objective optimization problem. These solutions are also called Pareto optimal solutions. Specifically, each Pareto solution is not inferior to any other solution on all objectives. One major merit for the use of trade-off analysis is that management can make the decision within the set of non-dominated solutions instead of considering the full range of feasible solutions. By comparing different options in the set of non-dominated solutions, a rational compromise among various objectives can be made.

In our multi-objective optimization problems, two formulations are provided as shown in Formulation 1 and Formulation 2. For Formulation 1, maximizing reliability and minimizing the cost are expected to be achieved at the same time. It can be easily seen from Figure 7.1 that the non-dominated solutions are in the set of $\left[t_0^*, \infty\right)$. For Formulation 2, minimizing risk is added as another objective. Since the risk is a decreasing function over time as shown in equation (7.4), the non-dominated solutions are in the same set of $\left[t_0^*, \infty\right)$. After these non-dominated solutions are identified, management can check the options within the set of $\left[t_0^*, \infty\right)$, and compromise among different objectives.

7.3.2 Multi-Attribute Utility Theory

Different objectives are generally not in the same scale and unit. They may also conflict with each other. Therefore, all objectives can be hardly optimized simultaneously. In reality, a compromise among different objectives is to be made. Multi-attribute utility theory (MAUT) is a classical multi-objective optimization approach, which solves the multi-optimization problem by using weights and the single utility function (von Winterfeldt and Edwards, 1986). In particular, the use of single utility function for each attribute can convert each objective into the same scale from 0 to 1 with the same unit of utility. The utility value reveals the attractiveness of each attribute. On the other hand, different importance weights are allocated for each single utility function. Finally, the multi-attribute utility function is obtained, which is actually a weighted sum of single utility functions. Mathematically, the multi-attribute utility function is given by

$$U(d_1, d_2, \ldots d_n) = \sum_{i=1}^{n} w_i u(d_i), \tag{7.5}$$

where each attribute is denoted by d_i, i=1,2,…n, the attractiveness of each attribute is represented by the single utility function $u(d_i)$, and w_i's are the importance weights allocated for different single utility functions. By maximizing this multi-attribute function, the optimal solution is obtained. In reality, it means that the attractiveness of the conjoint outcome of attributes is maximized under this optimal solution.

One important step in MAUT is the elicitation of single utility function. To achieve this, the equally likely certainty equivalent (ELCE) method was developed as a standard approach in MAUT (Keeney and Raiffa, 1976; von Winterfeldt and Edwards, 1986). In this method, certainty equivalents are obtained for a few 50-50 lotteries. These certainty equivalents are actually some special points on the single utility curve. Based on these estimated points, the single utility function is determined by fitting these points. For more detailed discussions on this method, interested readers can refer to Keeney and Raiffa (1976), von Winterfeldt and Edwards (1986). However, in practice, it is generally assumed that management is risk neutral towards each attribute (Scholz and Tietje, 2002). Under this assumption, the single utility function is a linear function for each attribute. Management only needs to determine two particular points for each attribute, i.e., d_i^0 and d_i^1. These two points are the lowest requirement and the highest expectation from management for the attribute d_i, and the superscripts of them represent their corresponding utility values.

Another important step is the estimation of weighting factors. This is done by comparing a certain scenario and a lottery (Keeney and Raiffa, 1976; von Winterfeldt and Edwards, 1986). More specifically, the certain scenario contains one attribute at its best level and the other attributes at their worst levels; the lottery contains all attributes at their best levels with probability *p* and all attributes at their worst levels with probability 1-*p*. When management is indifferent with these two choices, the probability *p* is the weighting factor allocated for that attribute at its best level in the certain scenario.

In summary, MAUT solves the multi-objective optimization problem in a straightforward way. It is quite understandable by incorporating weights and single utility functions into the decision making process. Based on the management's own attitude, the optimal solution is obtained, under which the overall attractiveness of the conjoint outcome of attributes is maximized.

7.3.3 Physical Programming Method

Physical programming, as an effective and competitive approach in multi-objective optimization, was originally proposed by Messac (1996). In order to express decision maker's preference towards each criterion, four distinct soft class functions are used. Specifically, they are Class 1-S: smaller is better, i.e., minimization; Class 2-S: larger is better, i.e., maximization; Class 3-S: value is better, i.e., seek value; and Class 4-S: range is better, i.e., seek range.

It is worth noting that hard class functions are omitted here. This is because hard class functions are used to describe constraints, which are not in the context of our decision problems. In our decision problems, cost and risk is expected to be minimized, and reliability is to be maximized, only Class 1-S and Class 2-S will be used. Their qualitative meanings are described in Figure 7.2, where $\overline{g}_i$ is the class function of attribute *i*, *i*=1,2,…,*n*; g_i is the value of each attribute, and g_{ij}'s , *j*=1,2,…,5, are the boundary values separating decision maker's satisfaction level towards each attribute into six ranges. Taking Class 1-S for an example, the six ranges are defined as:

unacceptable range: $g_i \geq g_{i5}$;

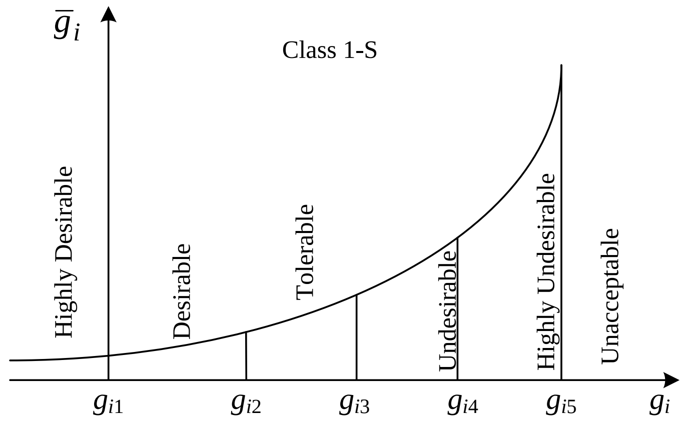
highly undesirable range: $g_{i4} \leq g_i \leq g_{i5}$;

undesirable range: $g_{i3} \leq g_i \leq g_{i4}$;

tolerable range: $g_{i2} \leq g_i \leq g_{i3}$;

desirable range: $g_{i1} \leq g_i \leq g_{i2}$;

highly desirable range: $g_i \leq g_{i1}$.



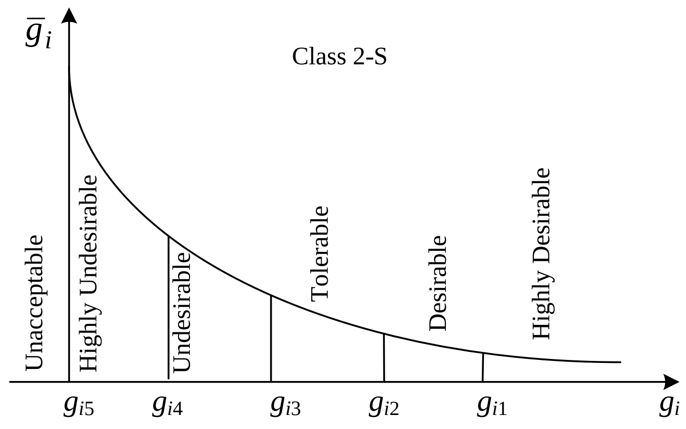


Figure 7.2 Qualitative meaning of Class 1-S and Class 2-S (Messac, 1996)

It can be seen that the decision maker's preference is deliberately described. In fact, the boundary values g_{ij}'s are the only parameters that the decision maker needs to specify. Based on them, the class function $\bar{g}_i$ can be determined following the standard procedure developed in Messac (1996), and interested readers can refer to it for more detailed discussions.

In fact, this standard procedure puts its most effort into the one versus others criteria rule (OVO rule), which expresses the preference regarding inter-criteria relationships. Specifically, suppose that we have the following two options: (1) full improvement of g_i across a given range, i.e., the tolerable range; (2) full improvement of all other criteria across the next better range, i.e. desirable range. Under the OVO rule, the first option is always preferred over the second one. It means that the worst performance has the highest priority to be improved. After these soft class functions are determined, the multi-objective decision model using physical programming approach is formulated as (Messac, 1996):

$$\begin{cases} \text{Minimize} & g(\boldsymbol{x}) = \log_{10}\left\{ \dfrac{1}{n_{sc}} \sum_{i=1}^{n} \bar{g}_i \left[g_i(\boldsymbol{x}) \right] \right\} \\ \text{Subject to} & g_i(\boldsymbol{x}) \le g_{i5} \quad \text{(for Class 1 - S)} \\ & g_i(\boldsymbol{x}) \ge g_{i5} \quad \text{(for Class 2 - S)} \\ & x_{jm} \le x_j \le x_{jM} \end{cases}$$

where n_{sc} is the number of soft class functions considered in the decision problem, x_{jm} and x_{jM} represent the minimum and maximum values of the corresponding decision variable x_j.

7.4 Numerical Examples

In this section, two numerical examples for software release time determination are provided. In particular, the first numerical example is to solve the release time problem under Formulation 1; and the second numerical example is for Formulation 2. Both two numerical examples consider the failure data set used in Pham and Zhang (1999), which is the failure account data in one hour intervals. In Pham and Zhang (1999), the software reliability model used is Goel-Okumoto (GO) model (Goel and Okumoto, 1979), whose mean value function and failure intensity function are given by

$$m(t) = a(1 - e^{-bt}) \text{ and } \lambda(t) = abe^{-bt}, \tag{7.6}$$

where *a* denotes the number of expected faults in the software, and *b* represents the fault detection rate. This model is also adopted in our analysis, and the estimated model parameters are given by $\hat{a} = 142.32$ and $\hat{b} = 0.1246$. In their work, parameters for the cost functions are also provided based on the real project data, and cost is measured in the unit of staff-units. Specifically, the parameter values are $c_0 = 50$, $c_1 = 700$, $c_2 = 60$, $c_3 = 3600$, $c_4 = 50000$, $t_w = 20$, $\mu_y = 0.1$, and $\mu_w = 0.5$. These parameter values can help us to quantify the cost function as shown in equation (7.3). While for the reliability function, we set *x*=1.

7.4.1 Example I

In the first numerical example, we only consider two attributes, i.e., reliability and cost. Based on the parameter values discussed above, reliability and cost can be quantified with ease based on (7.2) and (7.3). For the reliability and cost functions, their behavior is the same as shown in Figure 7.1. It can be calculated that cost is minimized at the time $t_0^* = 43.97$, under which the corresponding reliability and cost values are $R(t_0^*) = 0.9286$ and $C(t_0^*) = 30923$.

Results from Trade-Off Analysis

Trade off analysis considers the multi-objective optimization problem by using Pareto optimal solutions. In trade off analysis, each Pareto optimal solution is not inferior to any other solution on all objectives. In our problem, maximizing reliability and minimizing cost should be considered simultaneously. Since reliability is increasing over time and the expected cost is a convex function with its minimum value at the time $t_0^* = 43.97$, the Pareto optimal solutions in trade off analysis can be easily identified in the set of $[43.97, \infty)$. It can be seen that under Pareto optimal solutions, increasing reliability inevitably increases the expected software development cost as shown in Figure 7.3.

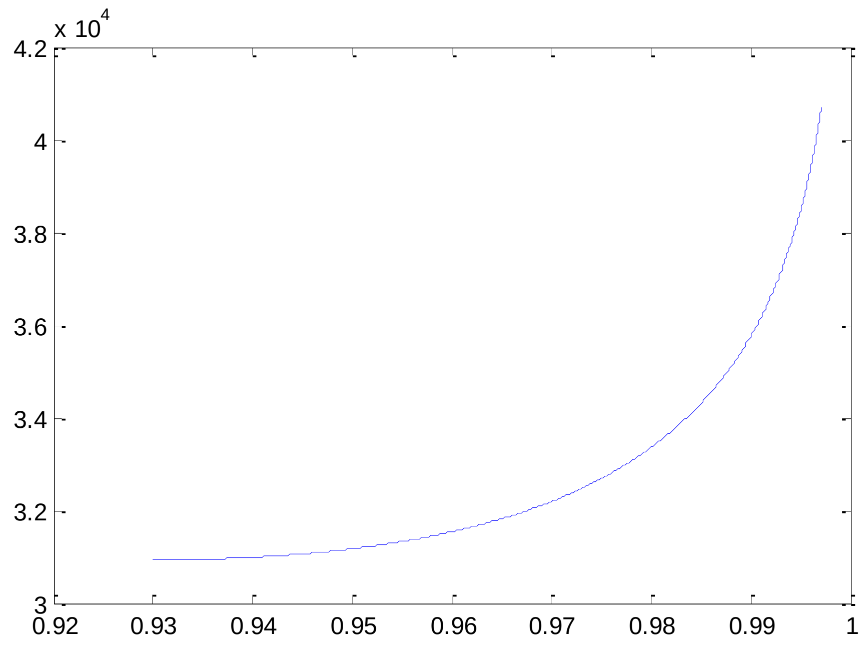


Figure 7.3 Non-dominated points of the consequence space with reliability and cost

In trade off analysis, Figure 7.3 provides management with a broader view of the decision problem, where the general trend of the non-dominated points of the consequence space is easily identified. Subsequently, management can select some typical points from the Pareto optimal solutions, and check their corresponding reliability and cost values. If management wants to increase the accuracy of the interval estimation, they can simply reduce the length of the interval. For instance, Table 7.1 provides numerical results based on the 5-hour interval estimation. Suppose that management prefers possible reliability-cost combinations from (0.9657, 31863) to (0.9814, 33589), then the interval estimation for the optimal release time is obtained as [50, 55]. If management wants to increase the accuracy of this estimation, they can further check numerical results based on the 1-hour interval estimation in the range of [50, 55]. This iterative process can be further conducted until satisfactory results are obtained.

Table 7.1 Numerical results based on the 5-hour interval estimation

t	$R(t)$	$C(t)$	$r(t)$
45	0.9370	30956	0.6573
50	0.9657	31863	0.2570
55	0.9814	33589	0.0436
60	0.9900	35761	0.0028
65	0.9946	38173	0.0001

Results from MAUT

MAUT solves the multi-optimization problem by using single utility functions and weights for each attribute. In our decision problem, we can first identify the single utility functions for reliability and cost respectively. As discussed before, risk neutrality is generally assumed, and management can use the linear single utility function (Scholz and Tietje, 2002). For the attribute reliability, management indicates that it can only accept reliability higher than 0.9, and the highest reliability expectation is 0.99. Therefore, we set $R^0 = 0.9$ and $R^1 = 0.99$, where the superscripts represent their corresponding utility values. Similarly, we receive two particular points for the attribute cost as $C^0 = 38000$ and $C^1 = 32000$. Based on the information above, the single utility function for reliability and cost are obtained as

$$u(R(t)) = \frac{R(t) - R^0}{R^1 - R^0} \text{ and } u(C(t)) = \frac{C^0 - C(t)}{C^0 - C^1}, \tag{7.7}$$

where $R(t)$ and $C(t)$ are used to represent $R(x|t)$ and $E[C(t)]$ for simplicity.

The following step is the estimation of weighting factors for each attribute. Management needs to compare the two choices as shown in Figure 7.4. When management is indifferent with these two choices, the probability *p* is the weighting factor w_R allocated for the attribute reliability. In our problem, management demonstrates that they put more importance on reliability. By comparing the two choices as shown in Figure 7.4, management is indifferent with them when *p* is equal to 0.7. Hence, $w_R = 0.7$ and $w_C = 0.3$.

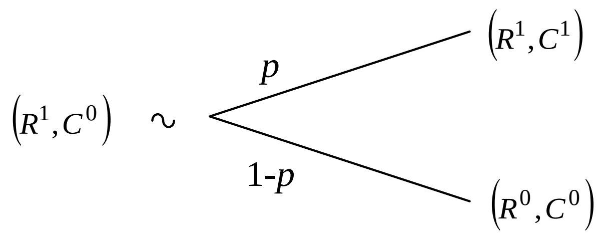


Figure 7.4 Two choices for the determination of the weighting factor for reliability

After the single utility functions and weights are identified, the multi-attribute utility function is obtained as

$$U(R(t), C(t)) = w_R u(R(t)) + w_C u(C(t)) . \tag{7.8}$$

By maximizing this multi-attribute utility function, the optimal release time is calculated as $T^* = 54.33$, under which corresponding reliability value and cost value are 0.9798 and 33325 respectively. In this example, numerical results from sensitivity analysis under the change of w_R are also provided as shown in Table 7.2. This is because the subjective assessment for the two choices as shown in Figure 7.4 may not be accurate enough. These results can help management to check whether the optimal solution is robust. Under the fact that management puts more importance on reliability, typical points of w_R around the predetermined value 0.7 are investigated. It can be seen that the maximum relative change of the optimal solution is within 10%. Therefore, the optimal solution obtained is acceptable. In addition, it also provides management with possible interval estimation that the optimal solution is in the range of [50.48, 60.00].

Table 7.2 Numerical results from sensitivity analysis under the change of w_R

w_R	t	$R(t)$	$C(t)$
0.9	60.00	0.9900	35760
0.8	57.70	0.9867	34722
0.7	54.33	0.9798	33325
0.6	51.91	0.9728	32450
0.5	50.48	0.9676	32000

Results from Physical Programming

Physical programming approach only needs management to provide the five boundary points for each attribute. In our decision problem, boundary points of class functions are shown in Table 7.3. In this example, since only reliability and cost are considered, data in the first two rows in Table 7.3 is used. In addition, for comparative purpose, we also set g_{i1} and g_{i5} equal to d_i^0 and d_i^1 used in MAUT. Following the standard procedure discussed in Messac (1996), the class functions for reliability and cost can be built as $\bar{g}_R[R(t)]$ and $\bar{g}_C[C(t)]$ respectively. The physical programming model for the decision problem considering reliability and cost is formulated as

$$\begin{cases} \text{Minimize} \quad g(t) = \log_{10}\left\{\dfrac{1}{2}\left(\bar{g}_R\left[R(t)\right] + \bar{g}_C\left[C(t)\right]\right)\right\} \\ \text{Subject to} \quad R(t) \ge 0.9, C(t) \le 38000, t > 0 \end{cases}.$$

Table 7.3 Boundary points of class functions

	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
reliability	0.99	0.97	0.94	0.91	0.9
cost	32000	34000	36000	37000	38000
risk	0.01	0.05	0.1	0.3	0.5

Since we only have one decision variable as the release time *t*, the optimization problem above can be solved easily by some software, i.e., MATLAB. The optimal release time is obtained as $T^* = 52.00$, under which corresponding reliability value and cost value are 0.9732 and 32481 respectively. It can be seen that under the optimal release time, both reliability and cost are in the desirable range ($g_{i1} \le g_i \le g_{i2}$).

Compared with MAUT, physical programming approach is easier for management to update the optimal solution. This is because weighting process as shown in Figure 7.4 is completely eliminated in the decision process. If management wants put more emphasis on reliability criterion, they can simply change the boundary points for reliability, and this process seems to be more meaningful.

7.4.2 Example II

Customers usually have a reliability requirement R_0, and management needs to try its best to make sure that this reliability requirement is satisfied. Generally, based on the estimated model parameters, the release time under this reliability requirement is calculated as $\hat{T}$. However, parameters are unknown in nature, and there exists the risk that software reliability cannot meet its reliability requirement due to parameter uncertainty (Xie et al., 2010). Therefore, in this numerical example, besides maximizing reliability and minimizing the software development cost, minimizing the risk that software cannot meet its reliability requirement is incorporated as well.

The failure data set used in Pham and Zhang (1999) is also adopted in this example. Therefore, parameter values in the previous example are used in this example as well. In addition, we set reliability requirement R_0 equal to 0.95. Hence, based on the GO model, the mean value of release time under this reliability target is obtained as $\hat{T} = 46.92$. Furthermore, following the standard procedure discussed in Xie et al. (2010), the variance of release time is calculated as $Var\left(\hat{T}\right) = 22.35$. Based on the above calculations, risk can be quantified with equation (7.4).

Results from Trade-Off Analysis

In this numerical example, maximizing reliability, minimizing cost and risk should be considered simultaneously. Since reliability is increasing over time, risk is decreasing over time, and the expected cost is a convex function with its minimum value at the time $t_0^* = 43.97$, the Pareto optimal solutions is still in the set of $[43.97, \infty)$ as the previous example. Under these Pareto optimal solutions, non-dominated points of the consequence space with reliability, cost and risk are shown in Figure 7.5. Similarly, Figure 7.5 provides management a broad view of the decision problem.

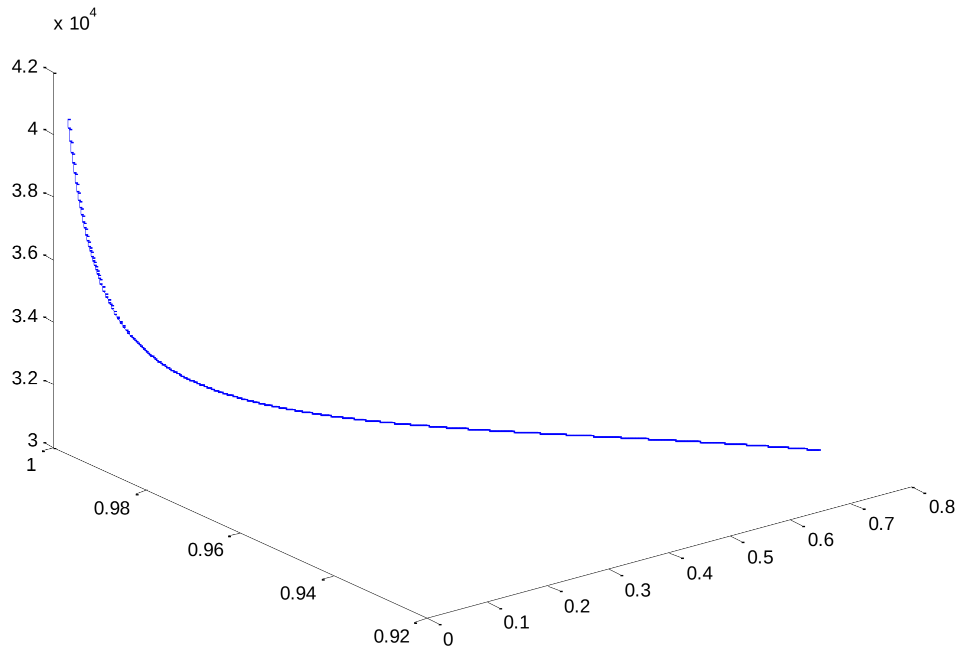


Figure 7.5 Non-dominated points of the consequence space with reliability, cost and risk

Subsequently, management can select some typical points for analysis, and Table 7.2 provides the numerical results based on the 5-hour interval estimation. However, in this numerical example, management needs to identify their preference towards three dimensional combinations, i.e. reliability-cost-risk combinations. This will certainly increase the complexity of the decision problem. Suppose that management prefers the reliability-cost-risk combinations from (0.9814, 33589, 0.0436) to (0.9900, 35761, 0.0028), then the interval estimation for optimal release time is obtained as [55, 60]. If management desires to increase the accuracy of the estimation, they can also try the 1-

hour interval estimation in the range of [55, 60]. However, to achieve this, management will spend more time and effort due to the increase of the dimension.

Results from MAUT

In MAUT, single utility functions and weights for each attribute should be determined first. In this example, risk is introduced as another dimension that management needs to consider. Management has indicated that they can only accept the risk to be lower than 0.5, and the highest expectation is 0.01. Therefore, we set the best and the worst value for risk as $r^0 = 0.5$ and $r^1 = 0.01$. The single utility function for risk is obtained as

$$u(r(t)) = \frac{r^0 - r(t)}{r^0 - r^1}. \tag{7.9}$$

The next step is the determination of weighting factors for each attribute. Since there are more dimensions to be considered in this decision problem, management probably wants to avoid answering the artificial lottery related questions. In this case, management can simply allocate equal importance for each attribute. In our problem, there are three attributes to be considered. Therefore, $w_R = w_C = w_r = 1/3$, and the multi-attribute utility function is given by

$$U\big(R(t), C(t), r(t)\big) = w_R u(R(t)) + w_C u(C(t)) + w_r u(r(t)). \tag{7.10}$$

The optimal solution on release time is obtained by maximizing this multi-attribute utility function, and it is calculated as $T^* = 54.94$, under which corresponding reliability, cost, and risk values are 0.981, 33567, and 0.0447 respectively. It can be seen that although the dimension of attributes is increased, the decision process can become even easier than the previous example if the equal importance allocation is assumed. This approach highly reduces the complexity of the decision process, and it can be very helpful for management when only some empirical results are needed.

Results from Physical Programming

Since minimizing the risk is considered as another objective in this example. All the data in Table 7.3 is used. The boundary points for risk attribute can help to construct the soft class function for it as $\bar{g}_r[r(t)]$, and the physical programming model in this example becomes

$$\begin{cases} \text{Minimize} \quad g(t) = \log_{10}\left\{\frac{1}{2}\left(\bar{g}_R[R(t)] + \bar{g}_C[C(t)] + \bar{g}_r[r(t)]\right)\right\} \\ \text{Subject to} \quad R(t) \geq 0.9, C(t) \leq 38000, r(t) \geq 0.5, t > 0 \end{cases}$$

By solving the optimization problem above, the optimal release time is obtained as $T^* = 56.01$, under which corresponding reliability, cost, and risk values are 0.9836, 34000 and 0.0272 respectively.

Obviously, compared with MAUT, management is no longer worried about answering many lottery related questions. The introduction of risk only requires management to specify five boundary points for it, and this is not a difficult task because the physical meanings of these points are quite clear. Although MAUT can simply adopt the equal importance allocation assumption, this may probably restrict the decision problem to a special case. Due to this consideration, when many objectives are to be compromised together, physical programming is generally better than MAUT.

7.5 Applicability and Limitations of Different Approaches

In this study, different multi-objective optimization approaches to software release time determination are investigated. Compared with previous single-objective optimization approach, they can describe management's attitude more accurately. A compromise among different objectives can be made by incorporating more information from management into the decision process. However, it should be noted that different multi-objective optimization approaches have their own properties, which imply the applicability and limitations of them.

Trade-off analysis can restrict the decision space into the Pareto optimal solutions. It provides management with the most information on the decision process. By comparing various combinations of objective values under non-dominated solutions, a compromise among different objectives can be gradually made. However, this decision process is essentially a trial process. Hence, it could be time-consuming and error-prone. This problem can become more serious when more than two objectives are considered. As shown in our second numerical example, it is not an easy task to compare the three-dimensional combinations of objectives. These combinations could possibly confuse the management. Therefore, trade-off analysis seems to be more helpful for management to get a broad view of the decision process. It is the most informative multi-objective optimization approach, and can help to identify the trend and change of the non-dominated points of the consequence space.

As to the MAUT, it is the most straightforward way to solve the multi-objective optimization problem. The use of the assumption of management's risk neutrality can greatly reduce the complexity the decision process in practice (Scholz and Tietje, 2002). If management can further demonstrate their equal importance weights allocation, the optimal solution of release time can be identified with the minimum complexity, and it can be updated with ease. In this case, the MAUT serves to be the best multi-objective optimization approach to software release time determination. However, these two assumptions may not reveal management's attitude in practice. In this case, management needs to answer some lottery related questions to obtain the single utility function and the weights. This process is quite tedious, and it is even more unexpected that answers to these lottery related questions may not be consistent over time. Thus, from this standpoint, the applicability of MAUT is restricted.

Compared with the MAUT, physical programming method completely eliminates the process of choosing weights. It only requires five boundary points from management for each attribute, such that six ranges are separated as shown in Figure 7.2. If management wants to set a more rigorous requirement for one attribute, they can simply change the corresponding five boundary points for this attribute, and the optimal solution is updated. However, it is worth noting that physical programming approach incorporates the OVO rule. It is assumed that the worst performance has the highest priory to be improved. Accordingly, under the optimal solution, different objectives are in the similar preference ranges. In our numerical examples, it can be seen that all objectives are in the desirable range. This special property of physical programming approach indicates its limitation in the case that some attributes are extremely more important than the others.

7.6 Conclusion

Most existing research formulates software release time determination problem as single-objective optimization problems, which have many disadvantages in the decision process. In fact, the optimal software release time problem is a multi-objective optimization problem, and a compromise among various objectives should be made based on the management's attitude. In this study, we propose two multi-objective optimization models as shown in Formulation 1 and Formulation 2. To solve these multi-objective optimization problems, different approaches are used including the trade off analysis, MAUT, and physical programming approach. These approaches are also compared based on the numerical examples. Applicability and limitations of various approaches are discussed in detail. These discussions can help management apply these methods in practice more appropriately.

Chapter 8 Conclusions

The objective of the research presented in this thesis was to improve software reliability analysis, and to study the corresponding release time determination problem by extending traditional software reliability models and decision models. This chapter summarizes the research results and highlights their significance. Limitations of current research and recommendations for future research are also presented.

8.1 Research Results and Contributions

For mathematical tractability and simplicity, some assumptions are made to facilitate the modeling of the software failure process. However, these assumptions may not be realistic in practice and the applicability of software reliability models is restricted. Therefore, relaxing these assumptions for both ASRMs and DDSRMs is of considerable importance.

An extension on ASRMs was presented in Chapter 3. This chapter introduced the modeling framework for open source software reliability and discussed the corresponding version-updating problem. It was found that traditional non-homogeneous Poisson process (NHPP) ASRMs underestimate the reliability of open source software. This is because these traditional models cannot describe the hump-shaped failure detection rate function properly. It was also found that for open source software, cost is no longer a major concern for version-updating problem. Thus, a new decision model was developed based on multi-attribute utility theory (MAUT). This decision model can help management to make a more reasonable decision. Since traditional ASRMs and decision models are only focused on the study of closed source software/commercial software, the research in this chapter is one of the first attempts on reliability analysis and optimal version-updating for open source software.

Besides the research on ASRMs, improvement on DDSRMs was also carried out and presented in Chapter 4. The objective of this research was to relax the basic assumption in traditional DDSRMs, where the current failure is assumed to be correlated with the most recent consecutive failures. A generic DDSRM was developed by relaxing this unrealistic assumption. It was found that the proposed model can cater for various failure correlations and existing DDSRMs are special cases of the proposed model. Experimental results reveal that the prediction accuracy is greatly enhanced by the proposed model.

Developing models is not the ultimate goal of software reliability modeling. It is more important to apply these models to decision-making problems, and software release time determination is a typical application. In Chapter 5, sensitivity analysis of release time was investigated. We took a recent proposed model by Lin and Huang (2008) as an example and applied different approaches to conduct the sensitivity analysis. It was found that global sensitivity analysis is a better choice for the complex nonlinear model. Furthermore, the simulation results from global sensitivity analysis can help management make a judicious decision on when to release the software. The research in this chapter provides practitioners a better understanding of different approaches to sensitivity analysis in the context of software reliability analysis.

Sensitivity analysis can identify what the significant parameters are, and more attention can be paid to them for more accurate estimates. However, when other information or data are not available, the improvement for the estimation can be hardly done. In this case, it is very important to study the effect of parameter uncertainty on release time determination, and this was presented in Chapter 6. It was found that when the mean value is used, there is 50% chance that the software cannot meet its reliability requirement. This is because parameters are unknown in nature and they are estimated based on failure data. In order to reduce the risk that software cannot meet its reliability requirement, a risk-based approach was proposed for release time determination with delay cost considerations. The proposed approach provides management a broader view of release time determination problem.

Furthermore, for software release time determination problem, different formulations for it were examined in Chapter 7. It was found that formulating release time determination as single-objective optimization problems can hardly describe the management's attitude accurately. Hence, multi-objective optimization model were developed for release time determination problem, and various multi-optimization approaches are used. By comparing these different multi-optimization approaches, management can apply these methods more appropriately for release time determination problem.

8.2 Future Research

Open source software (OSS) provides a new paradigm for software development. In this thesis, reliability analysis and optimal version updating for it was investigated. It should be noted that our proposed model is essentially an extension of traditional ASRMs. It is still an open question whether new methodology should be developed for describing the failure process of open source software. This is because there is a lot of information about software attributes available, and these data, if used properly, should greatly enhance the reliability analysis for open source software. Future research on this problem could possibly pave the way for a new stage of software reliability engineering.

As to the proposed generic data-driven software reliability model, it is worth noting that the great enhancement of prediction accuracy was achieved at the cost of spending more time for calculation. Although a hybrid generic algorithm was proposed to speed up the time for convergence, future research on more advanced algorithms will be useful. In addition, it will be also interesting to investigate the relationship between current failure and other software attributes. In this case, not only the failure history will be analyzed, but also the change of software attributes over time will be studied together.

Thirdly, for sensitivity analysis of release time, different approaches of it can have different limitations as discussed in detail in Chapter 5. These limitations require the users to apply different approaches properly in practice. Furthermore, although it has been shown that global sensitivity analysis is a better choice for complex nonlinear model, the procedure of implementing the analysis could be still difficult and time-consuming for practitioners. Therefore, developing a software tool with user-friendly interface may be necessary in the near future.

Finally, for software release time determination, although some decision models were developed and different multi-objective approaches were compared, no single model can be regarded as a universal model to suit all decision processes. Beyond the studies explored in this research, other approaches can be studied as well in the future, and extensions can be made by considering the specific properties of the decision process in practice.

References

Benke, K.K., Lowell, K.E. and Hamilton, A.J. (2008) 'Parameter uncertainty, sensitivity analysis and prediction error in a water-balance hydrological model', *Mathematical and Computer Modelling*, 47 (11-12), 1134-1149.

Ben-Tal, A. and Nemirovski, A. (2002) 'Robust optimization-methodology and applications', *Mathematical Programming*, 92 (3), 453-480.

Boland, P.J. and Chuiv, N.N. (2007) 'Optimal times for software release when repair is imperfect', *Statistics & Probability Letters*, 77 (12), 1176–1184.

Boland, P. J. and Singh, H. (2003) 'A birth-process approach to moranda's geometric software-reliability model', *IEEE Transaction on reliability*, 52 (2), 168-174.

Box, G.E.P., Hunter, W.G. and Hunter, S.J. *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building*, Wiley, New York, 1978.

Brito, A.J. and de Almeida, A.T. (2009) 'Multi-attribute risk assessment for risk ranking of natural gas pipelines', *Reliability Engineering & System Safety*, 94 (2), 187-198.

Bustamante, A.S. and Bustamante B.S. (2003) 'Multinomial-exponential reliability function: a software reliability model', *Reliability Engineering & System Safety*, 79 (3), 281-288.

Cai, K.Y., Cai, L., Wang, W.D., Yu, Z.Y. and Zhang, D. (2001) 'On the neural network approach in software reliability modeling', *Journal of Systems and Software*, 58 (1), 47-62.

Chang, Y.C. and Liu, C.T. (2009) 'A generalized JM model with applications to imperfect debugging in software reliability', *Applied Mathematical Modelling*, 33, 3578-3588.

Charette, R.N. (2005) 'Why software fails', *IEEE Spectrum*, 42 (9), 42-49.

Chen, K.Y. (2007) 'Forecasting system reliability based on support vector regression with genetic algorithms', *Reliability Engineering & System Safety*, 92 (4), 423-432.

Chiu, K.C., Ho, J.W. and Huang, Y.S. (2009) 'Bayesian updating of optimal release time for software systems', *Software Quality Journal*, 17, 99-120.

Dai, Y.S., Xie, M., Long, Q. and Ng, S.H. (2007) 'Uncertainty analysis in software reliability modeling by Bayesian approach with maximum-entropy principle', *IEEE Transactions on Software Engineering*, 33, 781-795.

Dohi, T., Nishio, Y. and Osaki, S. (1999) 'Optimal software release scheduling based on artificial neural networks', *Annals of Software Engineering*, 8, 167-185.

Eaddy, M., Zimmermann, T., Sherwood, K.D., Garg, V., Murphy, G.C., Nagappan, N., Aho, A.V., (2008) 'Do crosscutting concerns cause defects?', *IEEE Transactions on Software Engineering*, 34 (4), 497-515.

Edwards, W. (1977) 'Use of multiattribute utility measurement for social decision making', In: Bell, D.E., Keeney, R.L. and Raiffa, H. Editors, *Conflicting Objectives in Decision*, Willey, New York, 247–276.

Farmer, P.C. (1987) 'Testing the robustness of multiattribute utility theory in an applied setting', *Decision Sciences*, 18 (2), 178–193.

Ferreira, R.J.P., de Almeida, A.T. and Cavalcante, C.A.V. (2009) 'A multi-criteria decision model to determine inspection intervals of condition monitoring based on delay time analysis', *Reliability Engineering & System Safety*, 94

(5), 905-912.

Feller, J., Fitgerald, B., Hissam, S. and Lakhani, K. *Perspectives on Free and Open Source Software*, MIT Press: Cambridge, MA, 2005.

Fishburn, P.C. *Utility Theory for Decision Making*, Wiley, New York, 1970.

Goel, A.L. and Okumoto, K. (1979) 'Time-dependent error-detection rate model for software reliability and other performance measures', *IEEE Transactions on Reliability*, R28, 206-211.

Goel, A.L. (1985) 'Software reliability models: assumptions, limitations, and applicability', *IEEE Transactions on Software Engineering*, SE11, 1141-1423.

Gokhale, S.S., Lyu, M.R. and Trivedi, K.S. (2004) 'Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain', *Software Quality Journal*, 12, 211–230.

Gokhale, S.S., Lyu, M.R. and Trivedi, K.S. (2006) 'Incorporating fault debugging activities into software reliability models: A simulation approach', *IEEE Transactions on Reliability*, 55 (2), 281–292.

Goldberg, D. *Generic Algorithms in Seach, Optimization, and Machine Learning. Reading*, MA: Addison-Wesley, 1989.

Gyimothy, T., Ferenc, R. and Siket, I. (2005) 'Empirical validation of object-oriented metrics on open source software for fault prediction', *IEEE Transactions on Software Engineering*, 31 (10), 897-910.

Helton, J.C. and Davis, F.J. (2003) 'Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems', *Reliability Engineering & Systems Safety*, 81, 23-69.

Hertel, G., Niedner, S. and Hermann, S. (2003) 'Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel', *Research Policy,* 32 (7), 1159-1177.

Ho, J.W., Fang, C.C. and Huang, Y.S. (2008) 'The determination of optimal software release times at different confidence levels with consideration of learning effects', *Software testing, verification and reliability,* 18 (4), 221-249.

Ho, S.L., Xie, M. and Goh, T.N. (2003) 'A study of the connectionist models for software reliability prediction', *Computer and Mathematics with Applications*, 46 (7), 1037-1045.

Hu, Q.P., Xie, M., Ng, S.H. and Levitin, G. (2007) 'Robust recurrent neural network modeling for software fault detection and correction prediction', *Reliability Engineering & System Safety*, 92 (3), 332-340.

Huang, C.Y. (2005) 'Performance analysis of software reliability growth models with testing-effort and change-point', *Journal of Systems and Software,* 76 (2), 181-194.

Huang, C.Y. and Huang, W.C. (2008) 'Software reliability analysis and measurement using finite and infinite server queueing models', *IEEE Transactions on Reliability*, 57 (1), 192-203.

Huang, C.Y. and Kuo, S.K. (2002) 'Analysis of incorporating logistic testing-effort function into software reliability modeling', *IEEE Transactions on Reliability*, 51 (3), 261-270.

Huang, C.Y., Kuo, S.K. and Lyu, M.R. (2007) 'An assessment of testing-effort dependent software reliability growth models', *IEEE Transactions on Reliability*, 56 (2), 198-211.

Huang, C.Y. and Lin, C.T. (2006) ‘Software reliability analysis by considering fault dependency and debugging time lag’, *IEEE Transactions on Reliability*, 55 (3), 436-450.

Huang, C.Y. and Lo, J.H. (2006) ‘Optimal resource allocation for cost and reliability of modular software systems in the testing phase’, *Journal of Systems and Software*, 79 (5), 653-664.

Huang, C.Y. and Lyu, M.R. (2005a) ‘Optimal release time for software systems considering cost, testing-effort, and test efficiency’, *IEEE Transactions on Reliability*, 54 (4), 583-591.

Huang, C.Y. and Lyu, M.R. (2005b) ‘Optimal testing resource allocation, and sensitivity analysis in software development’, *IEEE Transactions on Reliability*, 54 (4), 592-603.

Hwang, S. and Pham, H. (2009) ‘Quasi-renewal time-delay fault-removal consideration in software reliability modeling’, *IEEE Transactions on Systems Man and Cybernetics Part A - Systems and Humans*, 39 (1), 200-209.

Illes-Seifert, T. and Paech, B. (2010) ‘Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs’, *Information and Software Technology*, 52 (5), 539-558.

Jelinski, Z. and Moranda, P.B. (1972), ‘Software reliability research’, in *Statistical Computer Performance Evaluation*, ed. by Freiberger W., Academic Press, New York, 465-497.

Johnson, M. E. *Multivariate Statistical Simulation*, John Wiley & Sons, New York, 1987.

Kapur, P.K., Bardhan, A.K. and Yadavalli, V.S.S. (2007) ‘On allocation of resources during testing phase of a modular software’, *International Journal of Systems Science,* 38 (6), 493-499.

Kapur, P.K., Goswami, D.N., Bardhan, A. and Singh, O. (2008a) ‘Flexible software reliability growth model with testing effort dependent learning process’, *Applied Mathematical Modelling,* 32 (7), 1298-1307.

Kapur, P.K., Singh, V.B., Anand, S. and Yadavalli, V.S.S. (2008b) ‘Software reliability growth model with change-point and effort control using a power function of the testing time’, *International Journal of Production Research,* 46 (3), 771-787.

Karunanithi, N., Whitley, D. and Malaiya, Y.K. (1992) ‘Prediction of software reliability using connectionist models’, *IEEE Transactions on Software Engineering,* 18 (7), 563-74.

Kecman, V. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logics Models,* MIT Press, 2001.

Keeney, R.L. and Raiffa, H. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs,* Wiley, New York, 1976.

Koch, H.S. and Kubat, P. (1983) ‘Optimal release time of computer software’, *IEEE Transaction on Software Engineering,* 9 (3), 323-327.

Kozlov, D., Koskinen, J., Sakkinen, M. and Markkula, J. (2008) ‘Assessing maintainability change over multiple software releases’, *Journal of Software Maintenance and Evolution-Research and Practice,* 20, 31–58.

Kim, S., Whitehead, E.J. and Zhang, Y. (2008) ‘Classifying software changes: Clean or buggy?’, *IEEE Transactions on Software Engineering,* 34 (2), 181-196.

Leung, Y.W. (1992) ‘Optimum software release time with a given cost budget’, *Journal of Systems and Software*, 17 (3), 233-242.

Li, X., Xie M. and Ng, S.H. (2010) ‘Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points’, *Applied Mathematical Modelling*, 34, 3560-3570.

Lin, C.T. and Huang, C.Y. (2008) ‘Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models’, *Journal of Systems and Software*, 81 (6), 1025-1038.

Liu, C.T. and Chang, Y.C. (2007) ‘A reliability-constrained software release policy using a non-Gaussian Kalman filter model’, *Probability in the Engineering and Informational Sciences*, 21 (2), 301-314.

Lo, J.H., Huang, C.Y., Chen, I.Y., Kuo, S.Y. and Lyu, M.R. (2005) ‘Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure’, *Journal of Systems and Software*, 76 (1), 3-13.

Lo, J.S. and Huang, C.Y. (2006) ‘An integration of fault detection and correction processes in software reliability analysis’, *Journal of Systems and Software*, 79, 1312-1323.

Lutz, R.R. and Mikulski I.C. (2004) ‘Empirical analysis of safety critical anomalies during operation’, *IEEE Transactions on Software Engineering*, 30 (3), 172-180.

Lyu, M. *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1996.

Makowski, D., Naud, C., Jeuffroy, M.H., Barbottin, A. and Monod, H. (2006) ‘Global sensitivity analysis for calculating the contribution of genetic parameters to the variance of crop model prediction’, *Reliability Engineering & System Safety*, 91 (10-11), 1142-1147.

Marcus, A., Poshyvanyk, D. and Ferenc, R. (2008) ‘Using the conceptual cohesion of classes for fault prediction in object-oriented systems’, *IEEE Transactions on Software Engineering*, 34 (2), 287-300.

Messac, A. (1996) ‘Physical programming: effective optimization for computational design’, *AIAA Journal*, 34 (1), 149-158.

Mockus, A., Fielding, R.T. and Herbsleb, J.D. (2002) ‘Two case studies of open source software development: Apache and Mozilla’, *ACM Transactions on Software Engineering and Methodology*, 11 (3), 309-346.

Montgomery, D.C. and Runger, G.C. *Applied Statistics and Probability for Engineers, 2nd ed.*, Wiley, New York, 1999.

Morali, N. and Soyer, R. (2003) ‘Optimal stopping in software testing’, *Naval Research Logistics*, 50 (1), 88-104.

Musa, J.D., Iannino, A. and Okumoto, K. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, 1987.

Musa, J.D. (2006) http://members.aol.com/JohnDMusa/users.htm

Nan, N. and Harter, D.E. (2009) ‘Impact of budget and schedule pressure on software development cycle time and effort’, *IEEE Transactions on Software Engineering*, 35 (5), 624-637.

Nelson, W. *Applied Life Data Analysis*, John Wiley & Sons, New York, 1982.

Ngo-The, A. and Ruhe, G. (2009) ‘Optimized resource allocation for software release planning’, *IEEE Transactions on Software Engineering*, 35 (1), 109-123.

Nishio, Y. and Dohi, T. (2003) ‘Determination of the optimal software release time based on proportional hazards software reliability growth models’, *Journal* *of Quality in Maintenance Engineering*, 9 (1), 48-65.

Ohba, M. (1984) ‘Software reliability analysis models’, *IBM Journal of Research and Development*, 28, 428-443.

Okumoto, K. and Goel, A.L. (1980) ‘Optimum release time for software systems, based on reliability and cost criteria’, *Journal of Systems and Software*, 1 (4), 315–318.

Pai, P.F. (2006) ‘System reliability forecasting by support vector machines with genetic algorithms’, *Mathematical and Computer Modeling*, 43 (3-4), 262-274.

Pai, P.F. and Hong, W.C. (2006) ‘Software reliability forecasting by support vector machines with simulated annealing algorithms’, *Journal of Systems and Software*, 79 (6), 747-755.

Park, C.S. *Contemporary Engineering Economics-4th Edition*, Pearson Education International, New Jersey, 2007.

Pham, H. *Software Reliability*, Springer-Verlag, Singapore, 2000.

Pham, H. (1996) ‘A software cost model with imperfect debugging, random life cycle and penalty cost’, *International Journal of Systems Science*, 27 (5), 455-463.

Pham, H. (2003) ‘Software reliability and cost models: perspectives, comparison, and practice’, *European Journal of Operational Research*, 149 (3), 475-489.

Pham, L. and Pham, H. (2000) ‘Software reliability models with time-dependent hazard function based on Bayesian approach’, *IEEE Transactions on Systems, Man, and Cybernetics Part A*, 30 (1), 25-35.

Pham, H. and Zhang, X.M. (1999) ‘A software cost model with warranty and risk costs’, *IEEE Transactions on Computers*, 48 (1), 71-75.

Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by* *an Accidental Revolutionary*, O'Reilly, 2001.

Sahinidis, N.V. (2004) 'Optimization under uncertainty: state-of-the-art and opportunities', *Computers & Chemical Engineering,* 28 (6-7), 971-983.

Saltelli, A. (2002) 'Making best use of model evaluations to compute sensitivity indices', *Computer Physics Communications*, 145, 280-297.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M. and Tarantola, S. *Global sensitivity analysis: the primer,* John Wiley & Sons, Chichester, 2008.

Samoladas, I., Angelis, L. and Stamelos, I. (2010) 'Survival analysis on the duration of open source projects', *Information and Software Technology,* 52 (9), 902-922.

Shanthikumar, J.G. (1981) 'A general software reliability model for performance prediction', *Microelectronics and Reliability*, 21, 671-682.

Schick, G.J. and Wolverton, R.W. (1978) 'An analysis of competing software reliability models', *IEEE Transactions on Software Engineering,* 4, 104-120.

Schneidewind, N.F. (1975) 'Analysis of error processes in computer software', in *Proceedings of the International Conference on Reliable Software*, IEEE Computer Society Press: Los Alamitos, CA, 337–346.

Schneidewind, N.F. (2001) 'Modelling the fault correction process', in *Proceedings of the 12th International Symposium on Software Reliability Engineering,* IEEE Computer Society Press: Los Alamitos, CA, 185–190.

Scholkopf, B. and Smola, A.J. *Learning with Kernels*, MIT Press, 2002.

Scholz, R.W. and Tietje, O. *Embedded Case Study Methods: Integrating Quantitative and Qualitative Knowledge, Sage*, California, 2002.

Sgarbossa, F. and Pham, H. (2010) ‘A cost analysis of systems subject to random field environments and reliability’, *IEEE Transactions on Systems Man and Cybernetics, Part C-Applications and Reviews*, 40 (4), 429-437.

Sitte, R. (1999) ‘Comparison of software-reliability-growth predictions: Neural networks vs. parametric-recalibration’, *IEEE Transactions on Reliability*, 48 (3), 285-291.

Sobol, I.M. (2001) ‘Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates’, *Mathematics and Computers in Simulation*, 55 (1-3), 271-280.

Su, Y.S. and Huang, C.Y. (2007) ‘Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models’, *Journal of Systems and Software*, 80 (4), 606-615.

Taguchi, G., Chowdhury, S. and Wu, Y. *Taguchi’s Quality Engineering Handbook*, Wiley, Hoboken, 2005.

Tamura, Y. and Yamada, S. (2008) ‘A component-oriented reliability assessment method for open source software’, *International Journal of Reliability, Quality and Safety Engineering*, 15 (1), 33-53.

Tamura, Y. and Yamada, S. (2009) ‘Optimisation analysis for reliability assessment based on stochastic differential equation modelling for open source software’, *International Journal of Systems Science*, 40 (4), 429-438.

Tian, L. and Noore, A. (2005a) ‘Evolutionary neural network modeling for software cumulative failure time prediction’, *Reliability Engineering & System Safety*, 87 (1), 45-51.

Tian, L. and Noore, A. (2005b) ‘Dynamic software reliability prediction: an approach based on support vector machines’, *International Journal of Reliability, Quality and Safety Engineering*, 12 (4), 309-321.

Tsay, R.S. *Analysis of Financial Time Series*, John Wiley & Sons, 2002.

Valdés, J.J. and Bonham-Carter, G. (2006) ‘Time dependent neural network models for detecting changes of state in complex processes: Applications in earth sciences and astronomy’, *Neural Networks*, 19 (2), 196-207.

Valdés, J.J. and Mateescu, G. (2002) ‘Time series models discovery with similarity-based neuro-fuzzy networks and genetic algorithms: A parallel implementation’, in *Proceedings of the Third International Conference on Rough Sets and Current Trends in Computing* (*RSCTC2002*), Malvern, 279-288.

Vapnik, V.N. *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

Vapnik, V.N. (1999) ‘An overview of statistical learning theory’, *IEEE Transactions on Neural Networks*, 10 (5), 950-958.

Ven, K. and Mannaert, H. (2008) ‘Challenges and strategies in the use of open source software by independent software vendors’, *Information and Software Technology*, 50 (9-10), 991-1002.

Volkova, E., Iooss, B. and Dorpe, F.V. (2008) ‘Global sensitivity analysis for a numerical model of radionuclide migration from the RRC “Kurchatov Institute” radwaste disposal site’, *Stochastic Environmental Research and Risk Assessment*, 22 (1), 17-31.

von Winterfeldt, D. and Edwards, W. *Decision Analysis and Behavioral Research*, Cambridge University Press, Cambridge, UK, 1986.

Wood, A. (1996) ‘Predicting software reliability’, *IEEE Computer,* 29 (11), 66-77.

Wu, Y.P., Hu, Q.P., Xie, M. and Ng, S.H. (2007), ‘Modeling and analysis of software fault detection and correction process by considering time dependency’, *IEEE Transactions on Reliability,* 56 (4), 629-642.

Xie, M. (1990) ‘A Markov process model for software reliability analysis’, *Applied Stochastic Models and Data Analysis,* 6, 207-214.

Xie, M. *Software Reliability Modelling,* World Scientific Publisher, Singapore, 1991.

Xie, M. and Hong, G.Y. (1998) ‘A study of the sensitivity of software release time’, *Journal of Systems and Software,* 44 (2), 163-168.

Xie, M., Hong, G.Y. and Wohlin, C. (1999) ‘Software reliability prediction incorporating information from a similar project’, *Journal of Systems and Software,* 49, 43-48.

Xie, M., Hu, Q.P., Wu, Y.P. and Ng, S.H. (2007) ‘A study of the modeling and analysis of software fault-detection and fault-correction processes’, *Quality and Reliability Engineering International,* 23 (4), 459-470.

Xie, M., Li, X. and Ng, S.H. (2010) ‘Risk-based software release policy under parameter uncertainty’, *Journal of Risk and Reliability,* accepted for publication.

Xie, M. and Yang, B. (2003) ‘A study of the effect of imperfect debugging on software development cost’, *IEEE Transactions on Software Engineering,* 29 (5), 471-473.

Xie, M., Yang, B. and Gaudoin, O. (2004) ‘Sensitivity analysis in optimal software release time problems’, *Opsearch,* 41 (4), 250-263.

Xie, M. and Zhao, M. (1992) ‘The Schneidewind software reliability model revisited’, in *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, IEEE Computer Society Press: Los Alamitos, CA, 184–192.

Yamada, S., Hishitani, J. and Osaki, S. (1993) ‘Software-reliability growth with a Weibull test-effort: a model and application’, *IEEE Transactions on Reliability,* 42 (1), 100-106.

Yamada, S., Ohba, M. and Osaki, S. (1983) ‘S-shaped reliability growth modeling for software error detection’, *IEEE Transactions on Reliability*, R-32, 475-484.

Yamada, S., Ohba, M. and Osaki, S. (1984) ‘S-shaped software reliability growth models and their applications’, *IEEE Transactions on Reliability*, R-33, 289-292.

Yamada, S. and Osaki, S. (1985) ‘Cost-reliability optimal release policies for software systems’, *IEEE transactions on reliability,* R-34 (5), 422-424.

Yang, B. and Li, X. (2007) ‘A study on software reliability prediction based on support vector machines’, in *Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management* (*IEEM2007*), Singapore, 1176-1180.

Yang, B. and Xie, M. (2000) ‘A study of operational and testing reliability in software reliability analysis’, *Reliability Engineering & System Safety,* 70 (3), 323-329.

Yang, B., Hu, H. and Jia, L. (2008) ‘A study of uncertainty in software cost and its impact on optimal software release time’, *IEEE Transactions on Software Engineering*, 34 (6), 813-835.

Yang, B., Tan, F. and Huang, H.Z. (2007) ‘Data selection for support vector machine based software reliability models’, in *Proceedings of International*

Conference on Reliability Engineering and Safety Engineering (*INCRESE2007*), Udaipur, 299-307.

Ye, Z.S., Li, Z.Z. and Xie, M. (2010) ‘Some improvements on adaptive genetic algorithms for reliability-related applications’, *Reliability Engineering & System Safety*, 95 (2), 120-126.

Yu, W. and Harris, T.J. (2008) ‘Parameter uncertainty effects on variance-based sensitivity analysis’, *Reliability Engineering & System Safety*, 94 (2), 596-603.

Zhang, X. and Pham, H. (2000) ‘Comparisons of nonhomogeneous Poisson process software reliability models and its applications’, *International Journal of Systems Science*, 31 (9), 1115-1123.

Zhao, M. and Xie, M. (1993) ‘Robustness of optimum software release policies’, in: *Proceedings of International Symposium on Software Reliability Engineering*, 218-225.

Zhao, M. and Xie, M. (1996) ‘On maximum likelihood estimation for a general non-homogeneous Poisson process’, *Scandinavian Journal of Statistics*, 23, 597-607.