Statistical Defect Prediction Models for Software Quality Assurance

Yan Luo

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Quality Systems) at

Concordia University

Montréal, Québec, Canada

July 2007

© Yan Luo, 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-34446-0
Our file Notre référence
ISBN: 978-0-494-34446-0

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Statistical Defect Prediction Models for Software Quality Assurance

Yan Luo

Software defects entail a highly-significant cost penalty in lost productivity and post-release maintenance. Early defect prevention and removal techniques can substantially enhance the profit realized on software products. The motivation for software quality improvement is most often expressed in terms of increased customer satisfaction with higher product quality, or more generally, as a need to position SAP Inc as a leader in quality software development. Thus, knowledge about how many defects to expect in a software product at any given stage during its development process is a very valuable asset. The great challenge, however, is to devise efficient and reliable prediction models for software defects. The first problem addressed in this thesis is software reliability growth modeling. We introduce an anisotropic Laplace test statistic that not only takes into account the activity in the system but also the proportion of reliability growth within the model.

The major part of this thesis is devoted to statistical models that we have developed to predict software defects. We present a software defect prediction model using operating characteristic curves. The main idea behind our proposed technique is to use geometric insight in helping construct an efficient prediction method to reliably predict the number of failures at any given stage during the software development process. Our predictive approach uses the number of detected faults in the testing phase. Data from actual SAP projects is used to illustrate the much improved performance of the proposed method in comparison with existing prediction approaches.

Acknowledgements

I would like to express my deepest gratitude to my MASc thesis supervisor Dr. A. Ben Hamza, and to Mr. Torsten Bergander from SAP Labs Canada in Montreal for their continuous support and invaluable suggestions throughout the supervision of my thesis. Their great help is essential to the completion of this thesis, more importantly, the challenging research that lies behind it.

First, I would like to thank SAP Inc. for sponsoring the High-Profile Research Alliance Project with Concordia University, and for the financial support during my one-year internship at SAP Labs Canada. It was a great learning experience and I fully enjoyed working so closely with the team of Mr. Bergander. I was very happy and proud to be part of his team and be given the chance to work on a challenging real-world industrial research project. Next, I would like to thank the SAP Research Lab manager Ms. Nolwen Mahé for her help throughout my internship and for supporting my work in many aspects. Finally, I am grateful to my colleagues and friends at Concordia University for many discussions on research and life in general, coffee breaks, and humorous moments.

Table of Contents

List of Figures

CHAPTER 1

Introduction

Software quality is becoming increasingly important. Software is now used in many demanding applications and software defects cost businesses and the software industry billions of dollars in lost productivity. Software quality impacts development costs, delivery schedules, and user satisfaction. While defects in financial or word-processing programs are annoying and very costly. When software-intensive systems fly airplanes, drive automobiles, control air traffic, run factories, or operate power plants, defects can cause serious damage and even physical harm. In spite of all its problems, software is ideally suited for critical applications: it does not wear out or deteriorate. Computerized control systems are so versatile, economical, and reliable that they are now the common choice for almost all systems. Software engineers must thus consider that their work could impact the health, safety, and welfare of many people.

Early defect prevention and removal techniques can substantially enhance the profit realized on software products. The motivation for software quality improvement is most often expressed in terms of increased customer satisfaction with higher product quality, or more generally, as a need to position SAP Inc as a leader in quality software development. Thus, knowledge about how many defects to expect in a software product at any given stage during its development process is a very

valuable asset. Being able to estimate the defects related cost driver characteristics will substantially improve the decision processes relevant for e.g. releasing a software product . In addition, the production process for software products can be substantially improved by employing a model that accounts for the dynamic nature of software production processes and reliably quantifies the expected defects that a future customer is likely to encounter.

The great challenge, however, is to devise efficient and reliable prediction models for software defects. The first problem addressed in this thesis is software reliability growth modeling. We introduce an anisotropic Laplace test statistic that not only takes into account the activity in the system but also the proportion of reliability growth within the model. This generalized approach is defined as a weighted combination of a growth reliability model and a non-growth reliability model.

The major part of this thesis is devoted to statistical models that we have developed to predict software defects for any software or firmware product. We present a software defect prediction model using operating characteristic curves. The main idea behind our proposed technique is to use geometric insight in helping construct an efficient and fast prediction method to accurately predict the cumulative number of failures at any given stage during the software development process. Our predictive approach uses the number of detected faults instead of the software failure-occurrence time in the testing phase. Data from actual SAP projects is used illustrate the effectiveness and the much improved performance of the proposed method in comparison with the Bayesian prediction approaches. Although additional investigations, such as a determination of post-release software defects, might provide a more detailed analysis of the predicted defects, the results presented in this provide compelling motivation for improved software quality.

1.1 Framework and Motivation

Software Quality Assurance (SQA) is defined as a planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes, and procedures. SQA includes the process of assuring that standards and procedures are established and are followed throughout the software acquisition life cycle. Compliance with agreed-upon standards and procedures is evaluated through process monitoring, product evaluation, and audits. Software development and control processes should include quality assurance approval points, where an SQA evaluation of the product may be done in relation to the applicable standards.

One of the many challenges faced when attempting to build a business case for software process improvement is the relative lack of credible measurement data. If a company does not have the data to build the business case, it does not have the improvement project to get the data. It is the classic chicken-and-egg dilemma. The motivation behind this thesis is to implement statistical models for predicting software defects using available failure data. The practitioners collect software defect data during software development processes but the decision support power of the collected data is wasted in most of the organizations. These defect data combined with the data of other features become a well-suited repository for using Bayesian statistics and machine learning techniques to predict future defects.

1.1.1 What are Defects?

A software engineer's job is to deliver quality products for their planned costs, and on their committed schedules. Software products must also meet the user's functional needs and reliably and consistently do the user's job. While the software functions are most important to the program's

users, these functions are not usable unless the software runs. To get the software to run reliably, however, engineers must remove almost all its defects. Thus, while there are many aspects to software quality, the first quality concern must necessarily be with its defects.

The reason defects are so important is that people make a lot of mistakes. In fact, even experienced programmers typically make a mistake for every seven to ten lines of code they develop. While they generally find and correct most of these defects when they compile and test their programs, they often still have a lot of defects in the finished product.

Some people mistakenly refer to software defects as bugs. When programs are widely used and are applied in ways that their designers did not anticipate, seemingly trivial mistakes can have unforeseeable consequences. As widely used software systems are enhanced to meet new needs, latent problems can be exposed and a trivial-seeming defect can truly become dangerous. While the vast majority of trivial defects have trivial consequences, a small percentage of seemingly silly mistakes can cause serious problems. Since there is no way to know which of these simple mistakes will have serious consequences, we must treat them all as potentially serious defects, not as trivial-seeming "bugs".

The term defect refers to something that is wrong with a program. It could be a misspelling, a punctuation mistake, or an incorrect program statement. Defects can be in programs, in designs, or even in the requirements, specifications, or other documentation. Defects can be redundant or extra statements, incorrect statements, or omitted program sections. A defect, in fact, is anything that detracts from the program's ability to completely and effectively meet the user's needs. A defect is thus an objective thing. It is something you can identify, describe, and count.

Simple coding mistakes can produce very destructive or hard-to-find defects. Conversely, many sophisticated design defects are often easy to find. The sophistication of the design mistake and the

impact of the resulting defect are thus largely independent. Even trivial implementation errors can cause serious system problems. This is particularly important since the source of most software defects is simple programmer oversights and mistakes. While design issues are always important, initially developed programs typically have few design defects compared to the number of simple oversights, typos, and goofs. To improve program quality, it is thus essential that engineers learn to manage all the defects they inject in their programs.

1.1.2 Software Reliability Growth Models

Software reliability engineering is focused on engineering techniques for developing and maintaining software systems whose reliability can be quantitatively evaluated. In order to estimate as well as to predict the reliability of software systems, failure data need to be properly measured by various means during software development and operational phases. Moreover, credible software reliability models are required to track underlying software failure processes for accurate reliability analysis and prediction.

Achieving highly reliable software from the customers perspective is a demanding job for all software engineers and reliability engineers. [24] summarizes the following four technical areas which are applicable to achieving reliable software systems, and they can also be regarded as four fault lifecycle techniques:

1. Fault prevention: to avoid, by construction, fault occurrences.

2. Fault removal: to detect, by verification and validation, the existence of faults and eliminate them.

3. Fault tolerance: to provide, by redundancy, service complying with the specification in spite

of faults having occurred or occurring.

4. Fault/failure forecasting: to estimate, by evaluation, the presence of faults and the occurrences and consequences of failures. This has been the main focus of software reliability modeling.

Fault prevention is the initial defensive mechanism against unreliability. A fault which is never created costs nothing to fix. Fault prevention is therefore the inherent objective of every software engineering methodology. Fault prevention mechanisms cannot guarantee avoidance of all software faults. When faults are injected into the software, fault removal is the next protective means. Two practical approaches for fault removal are software inspection and software testing, both of which have become standard industry practices in quality assurance.

When inherent faults remain undetected through the inspection and testing processes, they will stay with the software when it is released into the field. Fault tolerance is the last defending line in preventing faults from manifesting themselves as system failures. Fault tolerance is the survival attribute of software systems in terms of their ability to deliver continuous service to the customers. Software fault tolerance techniques enable software systems to (1) prevent dormant software faults from becoming active, such as defensive programming to check for input and output conditions and forbid illegal operations; (2) contain the manifested software errors within a confined boundary without further propagation, such as exception handling routines to treat unsuccessful operations; (3) recover software operations from erroneous conditions, such as checkpointing and rollback mechanisms; and (4) tolerate system-level faults methodically, such as employing design diversity in the software development. Finally if software failures are destined to occur, it is critical to estimate and predict them. Fault/failure forecasting involves formulation of the fault/failure relationship, an understanding of the operational environment, the establishment of software reliability

models, developing procedures and mechanisms for software reliability measurement, and analyzing and evaluating the measurement results. The ability to determine software reliability not only gives us guidance about software quality and when to stop testing, but also provides information for software maintenance needs.

As a major task of fault/failure forecasting, software reliability modeling has attracted much research attention in estimation (measuring the current state) as well as prediction (assessing the future state) of the reliability of a software system. A software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time. There are three main reliability modeling approaches: the error seeding and tagging approach, the data domain approach, and the time domain approach, which is considered to be the most popular one. The basic principle of time domain software reliability modeling is to perform curve fitting of observed time-based failure data by a pre-specified model formula, such that the model can be parameterized with statistical techniques (such as the Least Square or Maximum Likelihood methods). The model can then provide estimation of existing reliability or prediction of future reliability by extrapolation techniques. Software reliability models usually make a number of common assumptions, as follows. (1) The operation environment where the reliability is to be measured is the same as the testing environment in which the reliability model has been parameterized. (2) Once a failure occurs, the fault which causes the failure is immediately removed. (3) The fault removal process will not introduce new faults. (4) The number of faults inherent in the software and the way these faults manifest themselves to cause failures follow, at least in a statistical sense, certain mathematical formulae.

There are essentially two types of software reliability models:

- those that attempt to predict software reliability from design parameters

- those that attempt to predict software reliability from test data

The first type of models are usually called "defect density" models and use code characteristics such as lines of code, nesting of loops, external references, input/outputs, and so forth to estimate the number of defects in the software. The second type of models are often called software reliability growth models (SRGMs) since the number of faults (as well as the failure rate) of the software system reduces when the testing progresses, resulting in growth of reliability. These models attempt to statistically correlate defect detection data with known functions such as an exponential function. If the correlation is good, the known function can be used to predict future behavior. Software reliability growth models are the focus of Chapters 2 and 3.

Most software reliability growth models have a parameter that relates to the total number of defects contained in a set of code. If we know this parameter and the current number of defects discovered, we know how many defects remain in the code. Knowing the number of residual defects helps us decide whether or not the code is ready to ship and how much testing is required if we decide the code is not ready to ship. It gives us an estimate of the number of failures that customers will encounter when operating the software. The estimate helps us to plan the appropriate levels of support that will be required for defect correction after the software has shipped and determine the cost of supporting the software.

The most popular SRGMs are the so-called nonhomogeneous Poisson process (NHPP) models that have been widely used to track reliability improvement during software testing as well as to assess the software reliability, the number of remaining faults in the software, and the optimal software release schedule. These models enable software developers to evaluate software reliability in a quantitative manner, and have been successfully used to provide guidance in making decisions such as when to terminate testing the software or how to allocate available recourses.

1.1 Framework and Motivation

The general NHPP software reliability growth model is formulated based on the following assumptions:

- The occurrence of software faults follows an NHPP with mean value function $m(t)$ and failure intensity function $\lambda(t)$.
- The software fault intensity rate at any time is proportional to the number of remaining faults in the software at that time.
- When a software fault is detected, a debugging effort takes place immediately.

Let $\{N(t), t \geq 0\}$ denote a counting process representing the cumulative number of faults detected by the time t, and $m(t) = E[N(t)]$ denote its expectation. The failure intensity $\lambda(t)$ and the mean value functions of the software at time t are related as follows

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

and

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

The cumulative number of faults detected at time t follows a Poisson distribution with time-dependent mean value function as follows

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

The software reliability, i.e., the probability that no failures occur in $(s, s + t)$ given that the last failure occurred at testing time s is

$$R(t|s) = \exp[-(m(t + s) - m(t))]$$

The mean value function $m(t)$ is nondecreasing with respect to testing time t under the bounded condition $m(\infty) = a$, where a is the expected total number of faults to be eventually detected.

Model name	$m(t)$	$\lambda(t)$
Log-linear	$\frac{\exp(\alpha+\beta t)}{\beta}$	$\exp(\alpha+\beta t)$
Exponential (Goel-Okumoto)	$\alpha[1-\exp(-\beta t)]$	$\alpha\beta\exp(-\beta t)$
Weibull (Generalized Goel-Okumoto)	$\alpha[1-\exp(-\beta t^{\gamma})]$	$\alpha\beta\gamma t^{\gamma-1}\exp(-\beta t^{\gamma})$
Power law	$\left(\frac{t}{\alpha}\right)^{\beta}$	$\frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1}$
S-shaped	$\alpha[1-(1+\beta t)\exp(-\beta t)]$	$\alpha\beta^2 t\exp(-\beta t)$

Table 1.1: NHPP models.

Knowing its value can help us to determine whether the software is ready to be released to the customers and how much more testing resources are required. It can also provide an estimate of the number of failures that will eventually be encountered by the customers. The mean value function can be expressed as follows

$$m(t) = aF(t),$$

where $F(t)$ is the cumulative distribution function. Hence,

$$\lambda(t) = aF'(t) = [a - m(t)]\frac{F'(t)}{1-F(t)} = [a - m(t)]\rho(t),$$

where $\rho(t)$ is the failure occurrence rate per fault of the software, or the rate at which the individual faults manifest themselves as failures during testing. The quantity $[a - m(t)]$ denotes the expected number of faults remaining. The failure occurrence rate per fault (also known as *hazard* function)

$$\rho(t) = \frac{\lambda(t)}{m(\infty) - m(t)}$$

can be a constant, increasing, decreasing, or increasing/decreasing.

Table 4.1 and Figure 1.1 show examples of NHPP models with different failure intensity functions $\lambda(t;\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\alpha, \beta)$.

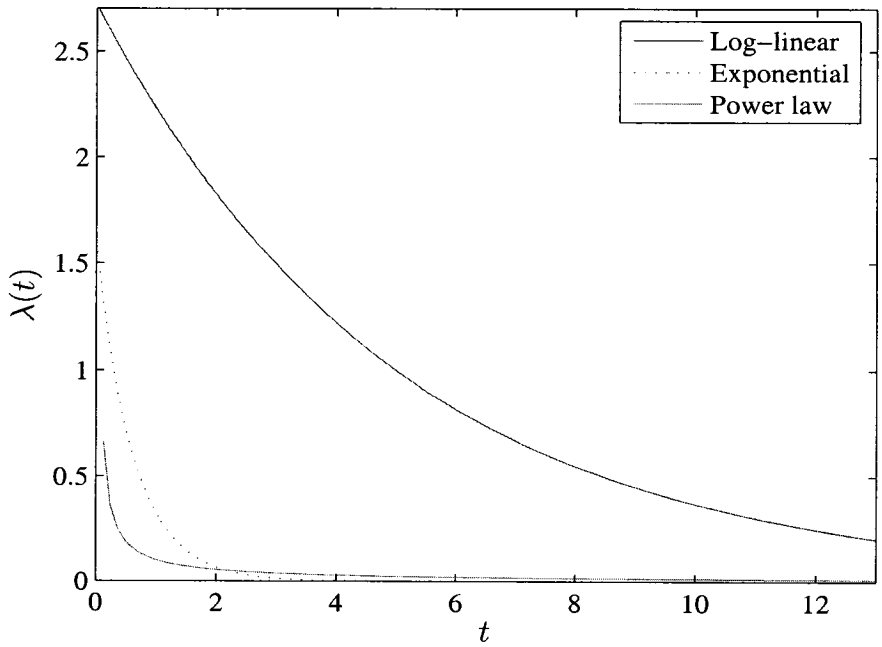


Figure 1.1: Illustration of failure intensity functions.

1.1.3 Operating Characteristic Curves

A statistical test provides a mechanism for making quantitative decisions about a process or processes. The intent is to determine whether there is enough evidence to "reject" a conjecture or hypothesis about the process. The conjecture is called the null hypothesis. Not rejecting may be a good result if we want to continue to act as if we "believe" the null hypothesis is true. Or it may be a disappointing result, possibly indicating we may not yet have enough data to "prove" something by rejecting the null hypothesis. A classic use of a statistical test occurs in process control studies, and it requires a pair of hypotheses:

H_0 : a null hypothesis

H_1 : an alternative hypothesis

The null hypothesis is a statement about a belief. We may doubt that the null hypothesis is true, which might be why we are "testing" it. The alternative hypothesis might, in fact, be what we believe to be true. The test procedure is constructed so that the risk of rejecting the null hypothesis, when it is in fact true, is small. This risk, α, is often referred to as the *significance level* of the test. By having a test with a small value of α, we feel that we have actually "proved" something when we reject the null hypothesis.

The risk of failing to reject the null hypothesis when it is in fact false is not chosen by the user but is determined, as one might expect, by the magnitude of the real discrepancy. This risk, β, is usually referred to as the *error of the second kind.* Large discrepancies between reality and the null hypothesis are easier to detect and lead to small errors of the second kind; while small discrepancies are more difficult to detect and lead to large errors of the second kind. Also the risk β increases as the risk α decreases. The risks of errors of the second kind are usually summarized by an *operating characteristic curve* (OC) for the test.

1.1.4 Bayesian Statistics

Bayesian inference is statistical inference in which evidence or observations are used to update or to newly infer the probability that a hypothesis may be true. The name "Bayesian" comes from the frequent use of Bayes' theorem in the inference process.

Bayesian inference uses aspects of the scientific method, which involves collecting evidence that is meant to be consistent or inconsistent with a given hypothesis. As evidence accumulates, the degree of belief in a hypothesis changes. With enough evidence, it will often become very high or very low. Thus, proponents of Bayesian inference say that it can be used to discriminate between conflicting hypotheses: hypotheses with a very high degree of belief should be accepted

as true and those with a very low degree of belief should be rejected as false. However, detractors say that this inference method may be biased due to initial beliefs that one needs to hold before any evidence is ever collected.

Bayesian inference uses a numerical estimate of the degree of belief in a hypothesis before evidence has been observed and calculates a numerical estimate of the degree of belief in the hypothesis after evidence has been observed. Bayesian inference usually relies on degrees of belief, or subjective probabilities, in the induction process and does not necessarily claim to provide an objective method of induction. Nonetheless, some Bayesian statisticians believe probabilities can have an objective value and therefore Bayesian inference can provide an objective method of induction.

Bayes' theorem adjusts probabilities given new evidence in the following way:

$$P(H_0|E) = \frac{P(E|H_0)P(H_0)}{P(E)},$$

where

- H_0 represents the null hypothesis that was inferred before new evidence, E, became available.
- $P(H_0)$ is called the prior probability of H_0.
- $P(E|H_0)$ is called the conditional probability of seeing the evidence E given that the hypothesis H_0 is true. It is also called the likelihood function when it is expressed as a function of H_0 given E.
- $P(E)$ is called the marginal probability of E: the probability of witnessing the new evidence E under all mutually exclusive hypotheses. It can be calculated as the sum of the product of all probabilities of mutually exclusive hypotheses and corresponding conditional probabilities: $\sum P(E|H_i)P(H_i)$.

- $P(H_0|E)$ is called the posterior probability of H_0 given E.

The factor $P(E|H_0)/P(E)$ represents the impact that the evidence has on the belief in the hypothesis. If it is likely that the evidence will be observed when the hypothesis under consideration is true, then this factor will be large. Multiplying the prior probability of the hypothesis by this factor would result in a large posterior probability of the hypothesis given the evidence. Under Bayesian inference, Bayes theorem therefore measures how much new evidence should alter a belief in a hypothesis.

Bayesian methods aim at assigning prior distributions to the parameters in the model in order to incorporate whatever *a priori* quantitative or qualitative knowledge we have available, and then to update these priors in the light of the data, yielding a posterior distribution via Bayes Theorem. The ability to include prior information in the model is not only an attractive pragmatic feature of the Bayesian approach, but it is also theoretically vital for guaranteeing coherent inferences.

1.2 Thesis Overview and Contributions

The organization of this thesis is as follows:

- The first Chapter contains a brief review of essential concepts and definitions which we will refer to throughout the thesis, and presents a short summary of material relevant to software defect prediction methods including software reliability growth models, Bayesian statistics, and operating characteristic curves.

- In Chapter 2, we present an anisotropic Laplace test statistic for software reliability growth enhancement. The proposed approach is defined as a nonlinear function of the Laplace trend test using redescending stabilization functions. The goal of stabilization functions is

to deal with the problem of increasing reliability growth by taking into account the absence of activity in the system. Experimental results with real software failure data illustrate the effectiveness and the much improved performance of the proposed method in software reliability.

- In Chapter 3, we introduce a new weighted Laplace test statistic for software reliability growth modelling. The proposed model not only takes into account the activity in the system but also the proportion of reliability growth within the model. This generalized approach is defined as a weighted combination of a growth reliability model and a non-growth reliability model. Experimental results illustrate the effectiveness and the much improved performance of the proposed method in software reliability modelling.

- In Chapter 4, we present a software defect prediction model using operating characteristic curves. The main idea behind our proposed technique is to use geometric insight in helping construct an efficient and fast prediction method to accurately predict the cumulative number of failures at any given stage during the software development process. Our predictive approach uses the number of detected faults instead of the software failure-occurrence time in the testing phase. Experimental results illustrate the effectiveness and the much improved performance of the proposed method in comparison with the Bayesian prediction approaches.

- In the **Conclusions** Chapter, we summarize the contributions of this thesis, and we propose several future research directions that are directly or indirectly related to the work performed in this thesis.

1.3 Publications

- Y. Luo, T. Bergander, and A. Ben Hamza, "Anisotropic Laplace trend to enhance software reliability growth modelling," *Proc. International Conference on Modelling and Simulation*, Montréal, Canada, May 2007.

- Y. Luo, T. Bergander, and A. Ben Hamza, "Software reliability growth modelling using a weighted Laplace test statistic," *Proc. IEEE International Computer Software and Applications Conference*, Beijing, China, July 2007.

- T. Bergander, Y. Luo, and A. Ben Hamza, "Software defects prediction using operating characteristic curves," *Proc. IEEE International Workshop on Software Stability at Work*, Las Vegas, USA, August 2007.

CHAPTER 2

Anisotropic Laplace Trend

We present an anisotropic Laplace test statistic for software reliability growth enhancement. The proposed approach is defined as a nonlinear function of the Laplace trend test using redescending stabilization functions. The goal of stabilization functions is to deal with the problem of increasing reliability growth by taking into account the absence of activity in the system. Experimental results with real software failure data illustrate the effectiveness and the much improved performance of the proposed method in software reliability.

2.1 Introduction

Software reliability engineering is centered on a key attribute, software reliability, which is defined as the probability of failure-free software operation for a specified period of time in a specified environment [25]. Among other attributes of software quality such as functionality, usability, capability, and maintainability, etc..., software reliability is generally accepted as the major factor in software quality since it quantifies software failures, which can make a powerful system inoperative. Software reliability engineering (SRE) is therefore defined as the quantitative study of

the operational behavior of software-based systems with respect to user requirements concerning reliability.

Existing SRE techniques suffer from a number of weaknesses. First of all, current SRE techniques collect the failure data during integration testing or system testing phases. Failure data collected during the late testing phase may be too late for fundamental design changes. Secondly, the failure data collected in the in-house testing may be limited, and they may not represent failures that would be uncovered under actual operational environment. This is especially true for high-quality software systems which require extensive and wide-ranging testing. The reliability estimation and prediction using the restricted testing data may cause accuracy problems. Thirdly, current SRE techniques or modeling methods are based on some unrealistic assumptions that make the reliability estimation too optimistic relative to real situations. Of course, the existing software reliability models have had their successes; but every model can find successful cases to justify its existence. Without cross-industry validation, the modeling exercise may become merely of intellectual interest and would not be widely adopted in industry. Thus, although SRE has been around for a while, credible software reliability techniques are still urgently needed, particularly for modern software systems [26].

Figure 2.1 shows an SRE framework in current practice [24]. First, a reliability objective is determined quantitatively from the customer's viewpoint to maximize customer satisfaction, and customer usage is defined by developing an operational profile. The software is then tested according to the operational profile, failure data collected, and reliability tracked during testing to determine the product release time. This activity may be repeated until a certain reliability level has been achieved. Reliability is also validated in the field to evaluate the reliability engineering efforts and to achieve future product and process improvements. It can be seen from Figure 2.1 that

there are four major components in this SRE process, namely (1) reliability objective, (2) operational profile, (3) reliability modeling and measurement, and (4) reliability validation. A reliability objective is the specification of the reliability goal of a product from the customer viewpoint. If a reliability objective has been specified by the customer, that reliability objective should be used. Otherwise, we can select the reliability measure which is the most intuitive and easily understood, and then determine the customer's "tolerance threshold" for system failures in terms of this reliability measure. The operational profile is a set of disjoint alternatives of system operational scenarios and their associated probabilities of occurrence. The construction of an operational profile encourages testers to select test cases according to the system's likely operational usage, which contributes to more accurate estimation of software reliability in the field.

Reliability modeling is an essential element of the reliability estimation process. It determines whether a product meets its reliability objective and is ready for release. One or more reliability models are employed to calculate, from failure data collected during system testing, various estimates of a product's reliability as a function of test time. Several interdependent estimates can be obtained to make equivalent statements about a product's reliability. These reliability estimates can provide the following information, which is useful for product quality management: (1) The reliability of the product at the end of system testing. (2) The amount of (additional) test time required to reach the product's reliability objective. (3) The reliability growth as a result of testing (e.g., the ratio of the value of the failure intensity at the start of testing to the value at the end of testing). (4) The predicted reliability beyond the system testing, such as the product's reliability in the field.

Despite the existence of a large number of models, the problem of model selection and application is manageable, as there are guidelines and statistical methods for selecting an appropriate

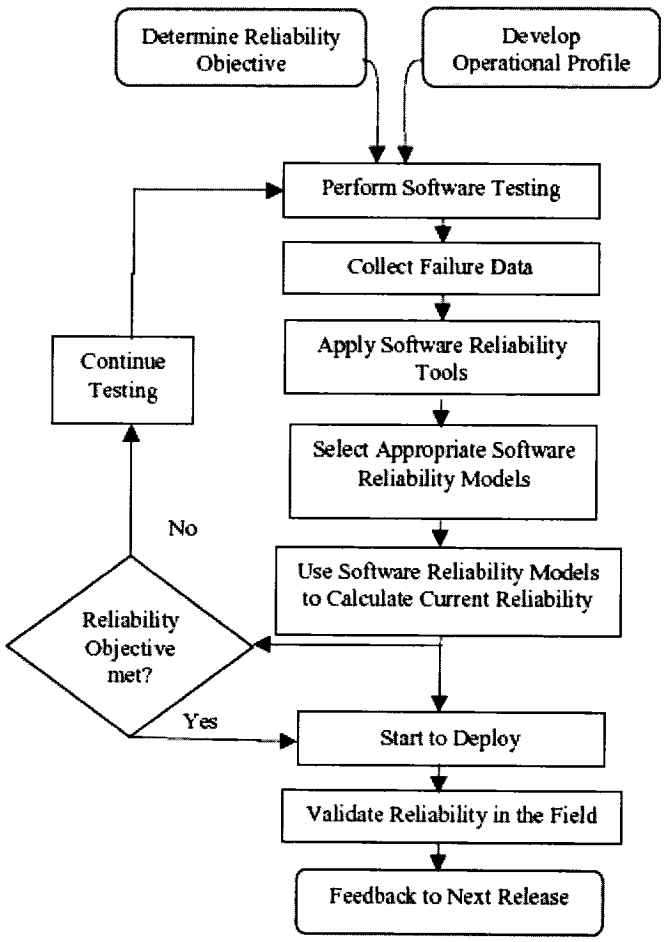


Figure 2.1: Software reliability engineering process overview.

model for each application. Furthermore, experience has shown that it is sufficient to consider only a dozen models, particularly when they are already implemented in software tools [24].

Using these statistical methods, "best" estimates of reliability are obtained during testing. These estimates are then used to project the reliability during field operation in order to determine whether the reliability objective has been met. This procedure is an iterative process, since more testing will be needed if the objective is not met. When the operational profile is not fully

developed, the application of a test compression factor can assist in estimating field reliability. A test compression factor is defined as the ratio of execution time required in the operational phase to execution time required in the test phase to cover the input space of the program. Since testers during testing are quickly searching through the input space for both normal and difficult execution conditions, while users during operation only execute the software with a regular pace, this factor represents the reduction of failure rate (or increase in reliability) during operation with respect to that observed during testing.

Finally, the projected field reliability has to be validated by comparing it with the observed field reliability. This validation not only establishes benchmarks and confidence levels of the reliability estimates, but also provides feedback to the SRE process for continuous improvement and better parameter tuning. When feedback is provided, SRE process enhancement comes naturally: the model validity is established, the growth of reliability is determined, and the test compression factor is refined.

During the development process of computer software systems, many software defects may be introduced and often lead to critical problems and complicated breakdowns of computer systems [1,2]. Hence, there is an increasing demand for controlling the software development process in terms of quality and reliability. Software reliability can be evaluated by the number of detected faults or the software failure-occurrence time in the testing phase which is the last phase of the development process, and it can be also estimated for the operational phase. A software failure is defined as an unacceptable departure of program operation caused by a software fault remaining in the software system [1,2,19].

It is, however, very difficult for developers to produce highly reliable software systems efficiently because of the diversified and complicated software requirements. Software reliability

models can provide quantitative measures of the reliability of software systems during software development processes [4, 5]. In recent years, several software reliability models have been proposed [6, 7]. In particular, software reliability models that describe software fault-detection or software failure-occurrence phenomena in the testing phase are referred to as software reliability growth models (SRGMs). The SRGMs have been proven to be successful in estimating the software reliability and the number of errors remaining in the software, and are very useful to assess the reliability for quality control and testing-process control of software development [4–9].

The rest of this chapter is organized as follows. In the next section, we formulate the problem and we briefly review the mathematical aspects of non-homogeneous Poisson processes. In Section 3, the likelihood function of the cumulative number of failures is derived. Motivated by the concept of a stabilization function as a growth-stopping criterion, we introduce in Section 4 our proposed anisotropic Laplace test statistic. Section 5 presents experimental results to demonstrate the much improved performance of the proposed approach in software reliability growth enhancement. Finally, we conclude in Section 6.

2.2 Non-homogeneous Poisson Process

Software failure data are usually available to the user in three basic forms:

1. in the form of a sequence of ordered failure times

   $0 < t_1 < t_2 < \ldots < t_n$

2. in the form of a sequence of interfailure times τ_i where $\tau_i = t_i - t_{i-1}$ for $i = 1, \ldots, n$

3. in the form of cumulative number of failures.

It is easy to verify that the failure and interfailure times are related by $t_i = \sum_{j=1}^{i} \tau_j$ as depicted in Figure 2.2.

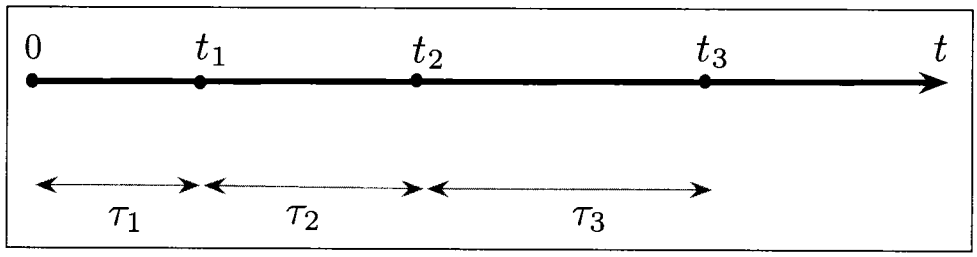


Figure 2.2: Illustration of failure times t_i and interfailure times τ_i.

The cumulative number of failures $N(t_i)$ detected by time t_i (i.e. the cumulative number of failures over the period $[0, t_i)$) defines a non-homogeneous Poisson process (NHPP) with failure intensity or rate function $\lambda(t_i)$ such that the rate function of the process is time-dependent. The mean value function $m(t_i) = E(N(t_i))$ of the process is given by $m(t_i) = \int_0^{t_i} \lambda(u)du$. Moreover, the function

$$f(t_i) = \lambda(t_i) \exp\left(-\int_0^{t_i} \lambda(u)du\right) = \lambda(t_i) \exp(-m(t_i))$$

defines a probability density function.

On the other hand, the number of failures $N(t_i, t_j)$ in any interval $[t_i, t_j)$ defines a non-homogeneous Poisson process with mean function

$$\int_{t_i}^{t_j} \lambda(u)du = m(t_j) - m(t_i).$$

That is,

$$\begin{aligned} &P(N(t_j) - N(t_i) = \kappa) \\ &\quad = \frac{(m(t_j) - m(t_i))^{\kappa}}{\kappa!} \exp(-(m(t_j) - m(t_i))). \end{aligned}$$

Software reliability $R(t_j|t_i)$ is defined as the probability that no software failure is detected in the time interval $(t_i, t_i + t_j)$, given that the last failure occurred at testing time t_i, and it is given by

$$R(t_j|t_i) = \exp\Big(-\big(m(t_i + t_j) - m(t_i)\big)\Big).$$

It is worth pointing out that if the failure intensity function is time-independent, then the cumulative number of failures $N(t_i)$ defines a homogeneous Poisson process (HPP).

2.3 The Likelihood Function

Assume we model the failure times using an NHPP with failure intensity function $\lambda(t;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is an unknown parameter vector.

Under the NHPP assumption, the failure times t_i define intervals for which only failure counts $n_i = N(i) - N(i-1)$ in the interval (t_{i-1}, t_i) are recorded, that is n_i is the number of failures during the i^{th} unit of time. Then, the probability of seeing n_i events in the interval (t_{i-1}, t_i) is given by

$$\begin{aligned} &P(N(t_{i-1}) - N(t_i) = n_i) \\ &\quad = \frac{(m(t_i) - m(t_{i-1}))^{n_i}}{n_i!} \exp(-(m(t_i) - m(t_{i-1}))). \end{aligned}$$

If we consider k time intervals, then the likelihood function is given by

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{k} \frac{(\int_{t_{i-1}}^{t_i} \lambda(u;\boldsymbol{\theta})du)^{n_i}}{n_i!} \exp\left(-\int_{t_{i-1}}^{t_i} \lambda(u;\boldsymbol{\theta})du\right) \\ &= \prod_{i=1}^{k} \frac{(m(t_i;\boldsymbol{\theta}) - m(t_{i-1};\boldsymbol{\theta}))^{n_i}}{n_i!} \exp(-m(t_n;\boldsymbol{\theta})), \end{aligned}$$

and the marginal probability that there are exactly $N = \sum_{i=1}^{k} n_i$ events is given by

$$G(\boldsymbol{\theta}) = \frac{(m(t_n))^N}{N!} \exp(-m(t_n))$$

Hence the conditional log-likelihood function is given by

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \log(L(\boldsymbol{\theta})) - \log(G(\boldsymbol{\theta})) \\ &= \sum_{i=1}^{k} n_i \log\Big(m(t_i;\boldsymbol{\theta}) - m(t_{i-1};\boldsymbol{\theta})\Big) \\ &\quad - N\log\Big(m(t_k;\boldsymbol{\theta})\Big) + C \\ &= \sum_{i=1}^{k} n_i \log\Big(\int_{t_{i-1}}^{t_i} \lambda(u;\boldsymbol{\theta})du\Big) \\ &\quad - N\log\Big(\int_{0}^{t_k} \lambda(u;\boldsymbol{\theta})du\Big) + C,\end{aligned}$$

where $C = \log(N!) - \log(\prod_{i=1}^{k} n_i!)$ is a constant.

2.4 Proposed Method

During testing and development of new systems, reliability trend analysis is needed to evaluate the progress of the development process [4, 5, 10, 11]. The hypotheses we wish to test are:

$$H_0 : \text{HPP}$$

$$H_1 : \text{NHPP}$$

where H_0 and H_1 are the null and the alternative hypotheses respectively.

Under the null hypothesis "H_0 : the process is a homogeneous Poisson process" (that is the intensity function is time independent), we define the Laplace trend as

$$U = \frac{\mathcal{L}(\theta_0)'}{E(-\mathcal{L}(\theta_0)'')},$$

where θ_0 is a component of the vector $\boldsymbol{\theta}$ such that its value makes the intensity function $\lambda(t;\boldsymbol{\theta})$ time independent.

2.4 Proposed Method

Assuming a type I error probability α, the Laplace trend values may be interpreted as follows:

- $U < -z_\alpha$: reliability growth
- $U > z_\alpha$: reliability deterioration (i.e. increasing failure intensity)
- $-z_\alpha < U < z_\alpha$: stable reliability,

where z_α is the is the upper α percentage of the standard normal distribution Z such that $P\{Z \geq z_\alpha\} = \alpha$ (i.e. z_α is the $100(1-\alpha)$ percentage point of the standard normal distribution).
If the null hypothesis H_0 : HPP is true, the distribution of the Laplace test statistic U is approximately normal $N(0,1)$. Consequently, if H_0 : HPP is true, the probability is $1-2\alpha$ that a value of the test statistic U falls between $-z_\alpha$ and z_α. Hence, we should reject H_0 if either $U > z_\alpha$ or $U < -z_\alpha$, and fail to reject H_0 if $-z_\alpha \leq U \leq z_\alpha$.

The objective of system reliability trend tests is to determine whether the pattern of failures is significantly changing with time. For example, when the occurrence of the events is an NHPP with a log-linear failure intensity function $\lambda(t) = \exp(\alpha + \beta t)$, then the null hypothesis may be expressed as $H_0 : \beta = 0$. Moreover, it can be shown that in the case of a log-linear failure intensity function [4,5,11], the Laplace test statistic is given by

$$U(k) = \frac{\sum_{i=1}^{k}(i-1)n_i - \frac{k-1}{2}\sum_{i=1}^{k} n_i}{\sqrt{\frac{k^2-1}{12}\sum_{i=1}^{k} n_i}}$$

The main limitation of the Laplace trend is that it does not take into account the presence or the absence of activity in the system. To circumvent this problem, we propose an anisotropic Laplace test statistic. The basic idea is to replace the Laplace trend factor $U(k)$ with an anisotropic Laplace

Function	$g(x)$
Green [12]	$\frac{\tanh(x)}{2x}$ (if $x \neq 0$)
Gaussian [13]	$\exp\left(-\frac{x^2}{2\sigma^2}\right)$
Lorentzian [13]	$\frac{1}{1+x^2/\sigma^2}$
Tukey's biweight	$\begin{cases} \frac{1}{2}\left(1-\left(\frac{x}{\sigma}\right)^2\right)^2 & \text{if } \lvert x\rvert \leq \sigma \\ 0 & \text{otherwise.} \end{cases}$
Huber minimax norm	$\begin{cases} \frac{1}{\sigma} & \text{if } \lvert x\rvert \leq \sigma \\ \frac{\text{sign}(x)}{x} & \text{otherwise.} \end{cases}$
Total Variation	$\frac{1}{2\lvert x\rvert}$ (if $x \neq 0$)
Hyper Surfaces	$\frac{1}{2\sqrt{1+x^2}}$
Geman & McClure	$\frac{1}{(1+x^2)^2}$

Table 2.1: reliability growth-stopping functions.

trend factor $A(k)$ which is defined as

$$A(k) = \begin{cases} g(U(k)) & \text{if no activity} \\ U(k) & \text{otherwise,} \end{cases}$$

where g is a "reliability growth-stopping" function as shown in Table 3.1 and Figure 2.3. The g-function is chosen to satisfy $g(x) \to 0$ when $x \to \infty$ so that the reliability growth is stopped when there is no activity in the system.

The parameter σ of the Gaussian and Lorentzian g-functions may be estimated using tools from

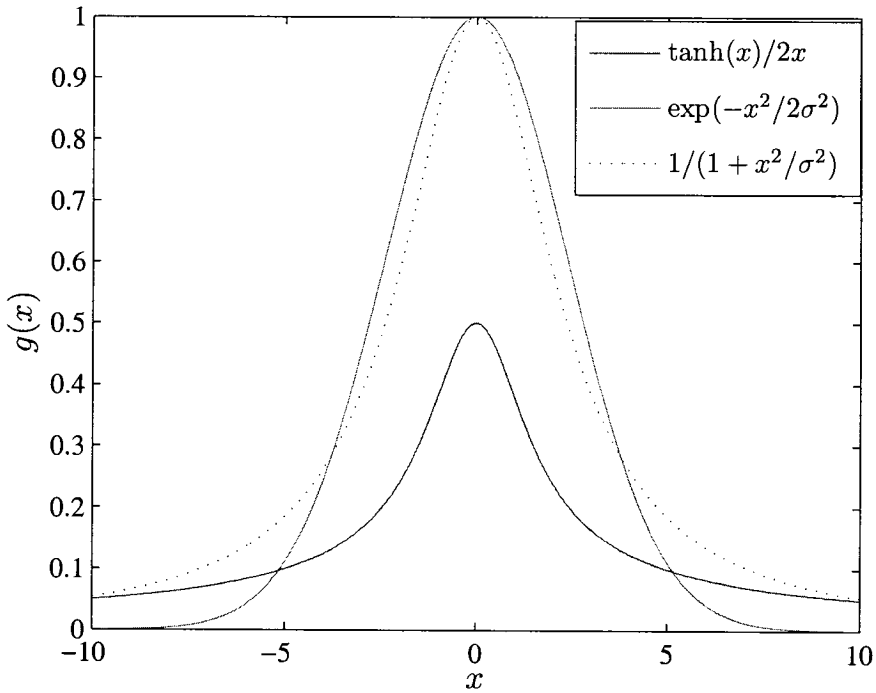


Figure 2.3: Plots of g-functions.

robust statistics as follows

$$\hat{\sigma} = 1.4826 \text{ MAD}\{(U(k) - U(k-1))_k\},$$

where MAD denotes the median absolute deviation [14], and the constant is derived from the fact that the MAD of a zero-mean normal distribution with unit variance is $0.6745 = 1/1.4826$.

2.5 Experimental Results

We tested our proposed anisotropic Laplace test statistic on a real software failure data which was taken from an SAP development system. The data contains daily software failures that was recorded for a period of 175 days. Moreover, there are no activities in the system during the test

phase process on the days 121, 122, 128, 142, 143, 144, 145, 146, 147, 148, 149 and 150. Figure 2.4 displays the scatter plot of cumulative failure number versus failure time, and it clearly illustrates an improving system since the probability of failures stabilizes substantially after a period of 150 days.

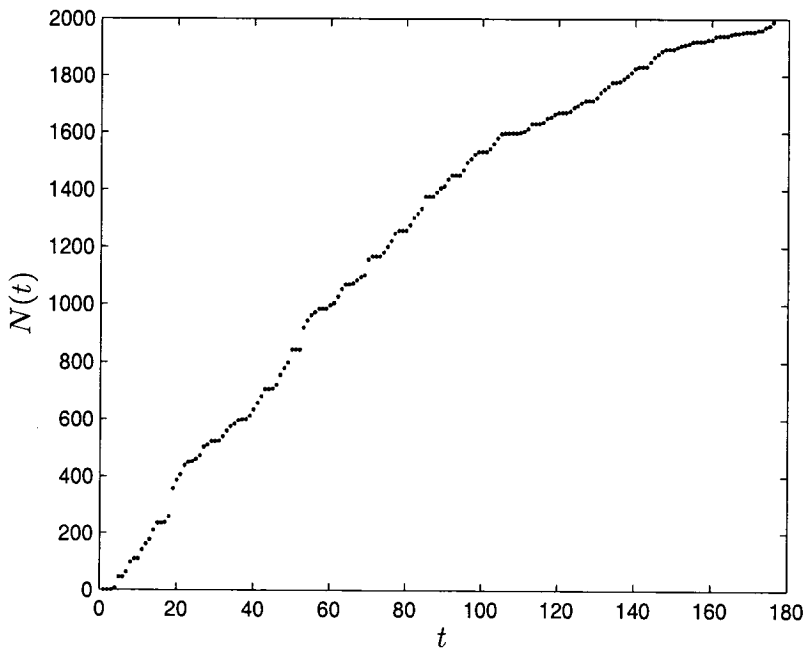


Figure 2.4: Cumulative Number of Failures vs. Failure Time.

2.5.1 Anisotropic Laplace trend results

Using a significance level of $\alpha = 5\%$, Figure 2.5 through Figure 2.7 show the anisotropic Laplace trends using the following reliability growth-stopping functions:

- **Green's function:**

$$g(U(k)) = \frac{\tanh(U(k) - U(k-1))}{2[U(k) - U(k-1)]}$$

- **Gaussian function:**

$$g(U(k)) = \exp\left(-\frac{(U(k) - U(k-1))^2}{2\sigma^2}\right)$$

- **Lorentzian function:**

$$g(U(k)) = \frac{1}{1 + \frac{(U(k) - U(k-1))^2}{\sigma^2}}$$

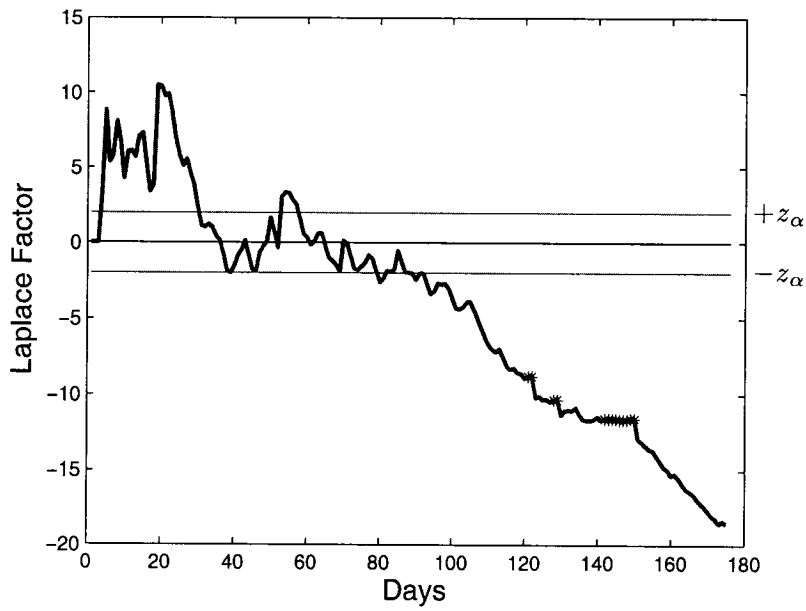


Figure 2.5: Anisotropic Laplace trend using Green's function.

We can see from Figures 2.5-2.7 that the portions of the anisotropic Laplace trends displayed as star-points clearly correct the original Laplace trend when no activity took place during the days 121, 122, 128, 129, 142, 143, 144, 145, 146, 147, 148, 149, 150. Furthermore, The experimental results clearly indicate that Green's function gives the best results.

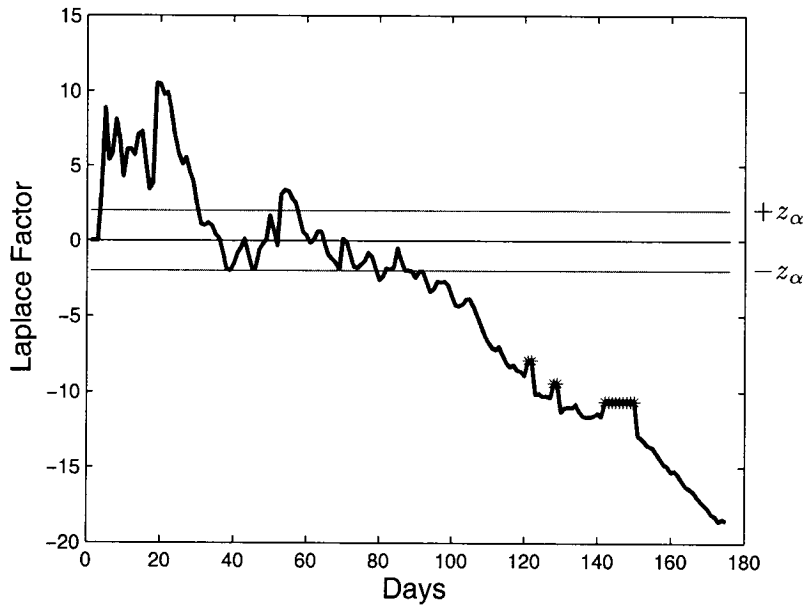


Figure 2.6: Anisotropic Laplace trend using Gaussian function.

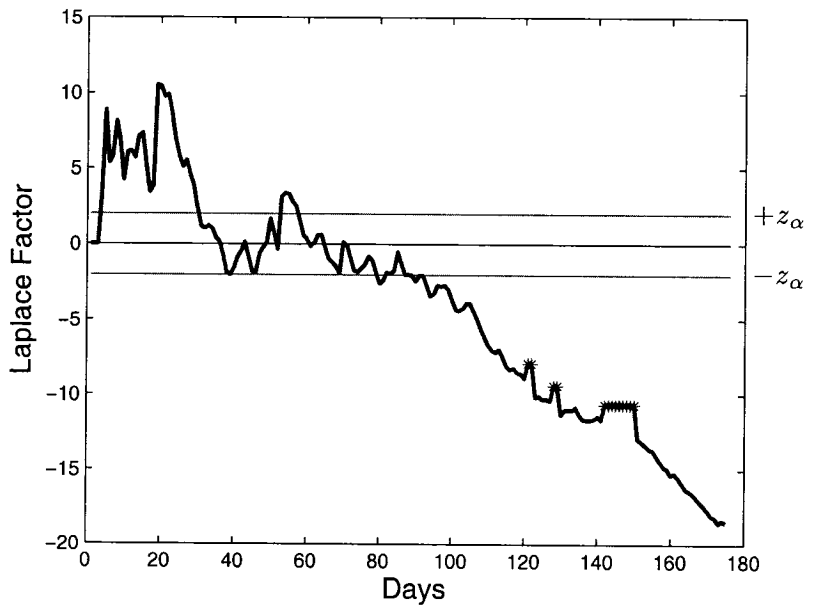


Figure 2.7: Anisotropic Laplace trend using Lorentzian function.

2.5.2 Adjusted Laplace trend results

The Laplace test statistic is a test for the null hypothesis H_0 that the data come from an HPP. Thus rejection of H_0 means that the process is not an HPP, but it could still in principle be a renewal process and hence still has no trend. In order to improve the test performance when the null hypothesis is a more general renewal process, the Lewis-Robinson (LR) test should be used [15]. The LR test is basically a scaled version of the Laplace test and it is defined as

$$U_{LR} = \frac{U(k)}{\widehat{CV}(\tau)},$$

where $\widehat{CV}(\tau)$ is an estimate of the coefficient of variation of the interfailure times τ_i, and it is calculated in terms of the mean and the standard deviation of interfailure times as follows

$$\widehat{CV}(\tau) = \frac{\sigma_\tau}{\bar{\tau}},$$

with τ representing the variable of interfailure times.

The reason for dividing the Laplace trend by the coefficient of variation is to account for non-exponential distributions of the interfailure times and also in order to insure that U_{LR} follows a standard normal distribution whenever the data come from a renewal process. Moreover, when the null-hypothesis is an renewal process model with non-exponential inter arrival times, this adjustment maintains the type-I error probability better than the Laplace test [15].

The adjusted Laplace trend is displayed in Figure 2.8 which clearly shows a growth reliability improvement over the Laplace trend.

We define the adjusted anisotropic Laplace trend factor $A_{LR}(k)$ as follows

$$A_{LR}(k) = \begin{cases} g(U_{LR}(k)) & \text{if no activity} \\ U_{LR}(k) & \text{otherwise.} \end{cases}$$

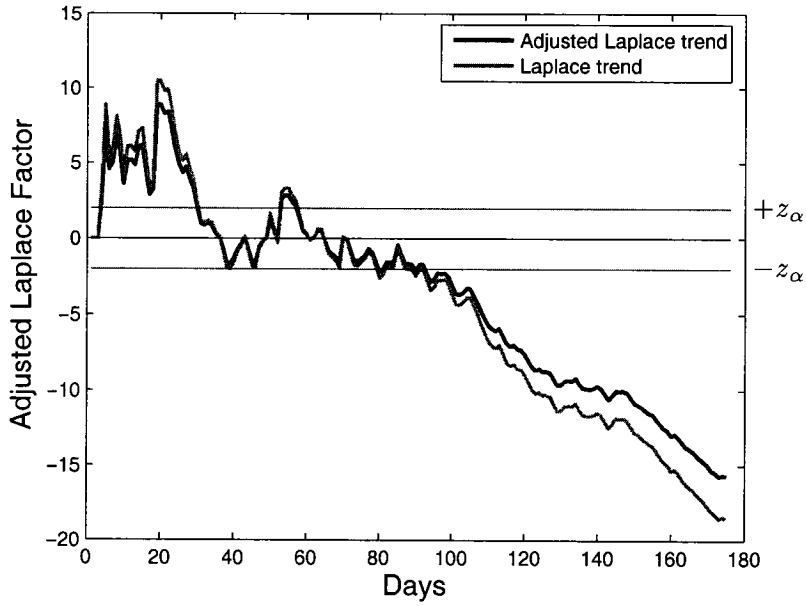


Figure 2.8: Adjusted Laplace trend.

Figure 2.9 through Figure 2.11 show the adjusted anisotropic Laplace trends using different "reliability growth-stopping" functions.

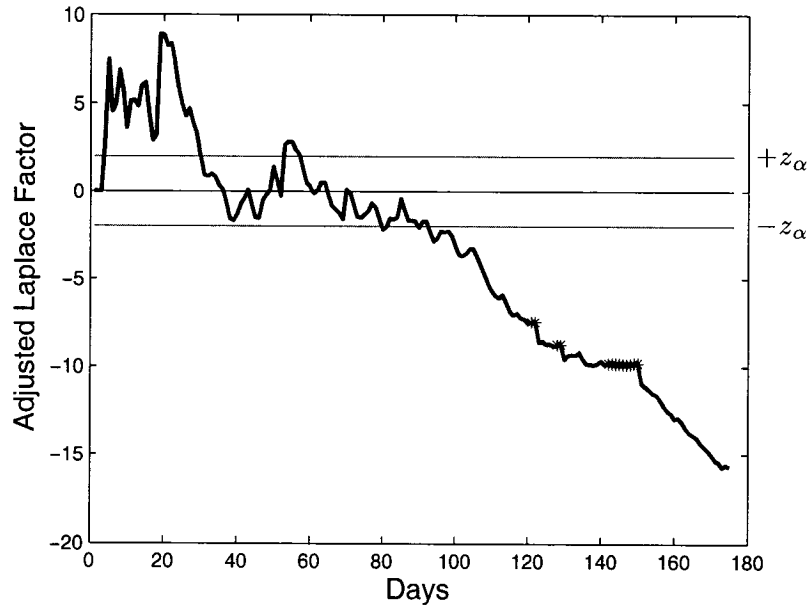


Figure 2.9: Adjusted anisotropic Laplace trend using Green's function.

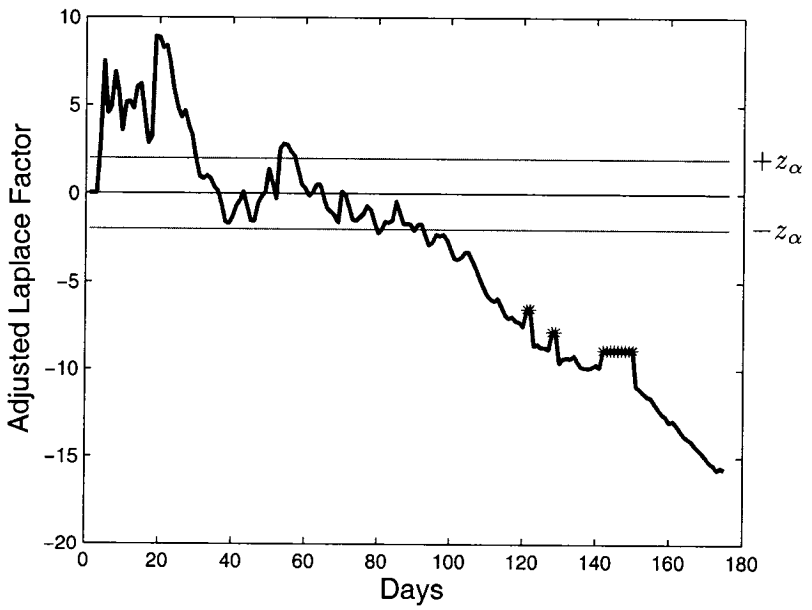


Figure 2.10: Adjusted anisotropic Laplace trend using Gaussian function.

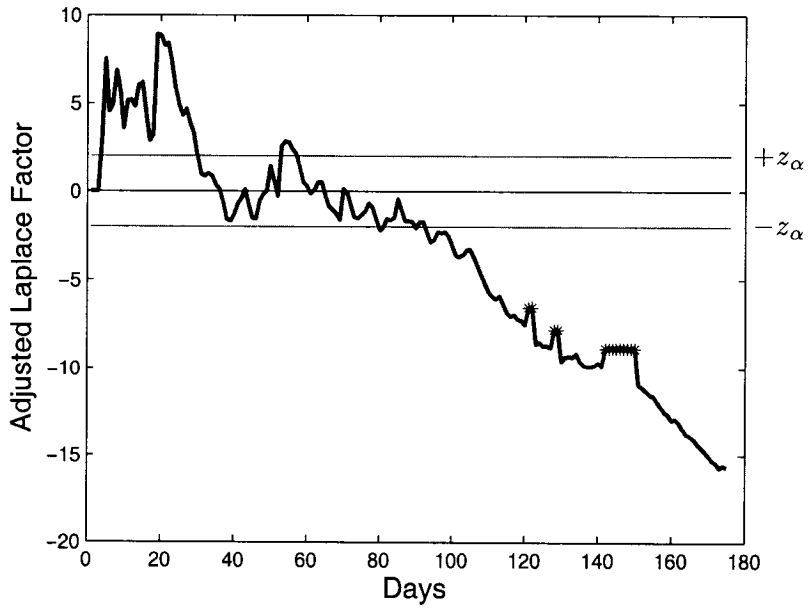


Figure 2.11: Adjusted anisotropic Laplace trend using Lorentzian function.

2.6 Conclusions

In this chapter, we proposed a nonlinear measure of trend for software reliability growth enhancement. The proposed test statistic is defined in terms of redescending stabilization functions which tackle the problem of increasing reliability growth by taking into account the activity in the system. The experimental results clearly indicate a much improved performance of the proposed anisotropic Laplace test statistic in software reliability growth enhancement.

CHAPTER 3

Weighted Anisotropic Laplace Test Statistic

We introduce a new weighted Laplace test statistic for software reliability growth modelling. The proposed model not only takes into account the activity in the system but also the proportion of reliability growth within the model. This generalized approach is defined as a weighted combination of a growth reliability model and a non-growth reliability model. Experimental results illustrate the effectiveness and the much improved performance of the proposed method in software reliability modelling.

3.1 Introduction

During the development process of computer software systems, many software defects may be introduced and often lead to critical problems and complicated breakdowns of computer systems [1,2]. Hence, there is an increasing demand for controlling the software development process in terms of quality and reliability. Software reliability can be evaluated by the number of detected faults or the software failure-occurrence time in the testing phase which is the last phase of the development process, and it can be also estimated for the operational phase. A software failure is

defined as an unacceptable departure of program operation caused by a software fault remaining in the software system [1, 2, 19].

It is, however, very difficult for developers to produce highly reliable software systems efficiently because of the diversified and complicated software requirements. Software reliability models can provide quantitative measures of the reliability of software systems during software development processes [4, 5]. In recent years, several software reliability models have been proposed [6, 7]. In particular, software reliability models that describe software fault-detection or software failure-occurrence phenomena in the testing phase are referred to as software reliability growth models (SRGMs). The SRGMs have been proven to be successful in estimating the software reliability and the number of errors remaining in the software, and are very useful to assess the reliability for quality control and testing-process control of software development [4–9].

The rest of this chapter is organized as follows. In the next section, we formulate the problem and we briefly review the mathematical aspects of non-homogeneous Poisson processes. In Section 3, the likelihood function of the cumulative number of failures is derived. In Section 4, we propose a weighted Laplace test statistic which is defined in terms of a weighted combination of a growth reliability model and a non-growth reliability model. Section 5 presents experimental results to demonstrate the much improved performance of the proposed approach in software reliability growth modelling. Finally, we conclude in Section 6.

3.2 Proposed Method

During testing and development of new systems, reliability trend analysis is needed to evaluate the progress of the development process [4, 5, 10, 11]. The hypotheses we wish to test are:

$$H_0 : \text{HPP}$$

$$H_1 : \text{NHPP}$$

where H_0 and H_1 are the null and the alternative hypotheses respectively.

Under the null hypothesis, "H_0 : the process is a homogeneous Poisson process" (that is the intensity function is time independent), we define the Laplace trend as

$$U = \frac{\mathcal{L}(\theta_0)'}{E(-\mathcal{L}(\theta_0)'')},$$

where θ_0 is a component of the vector $\boldsymbol{\theta}$ such that its value makes the intensity function $\lambda(t;\boldsymbol{\theta})$ time independent.

Assuming a type I error probability $\alpha = P\{\text{reject } H_0 | H_0 \text{ is true}\}$, the Laplace trend values may be interpreted as follows:

- $U < -z_\alpha$: reliability growth
- $U > z_\alpha$: reliability deterioration (i.e. increasing failure intensity)
- $-z_\alpha < U < z_\alpha$: stable reliability,

where z_α is the is the upper α percentage of the standard normal distribution Z such that $P\{Z \geq z_\alpha\} = \alpha$ (i.e. z_α is the $100(1-\alpha)$ percentage point of the standard normal distribution). If "H_0 : HPP" is true, the distribution of the Laplace test statistic U is approximately normal $N(0,1)$. Hence, if "H_0 : HPP" is true, the probability is $1 - 2\alpha$ that a value of the test statistic U falls between $-z_\alpha$ and z_α. Hence, we should reject H_0 if either $U > z_\alpha$ or $U < -z_\alpha$, and fail to reject H_0 if $-z_\alpha \leq U \leq z_\alpha$.

The objective of system reliability trend tests is to determine whether the pattern of failures is significantly changing with time. For example, when the occurrence of the events is an NHPP with a log-linear failure intensity function $\lambda(t) = \exp(\alpha + \beta t)$, then the null hypothesis may be expressed as $H_0 : \beta = 0$. Moreover, it can be shown that in the case of a log-linear failure intensity function [4, 5, 11], the Laplace test statistic is given by

$$U(k) = \frac{\sum_{i=1}^{k}(i-1)n_i - \frac{k-1}{2}\sum_{i=1}^{k} n_i}{\sqrt{\frac{k^2-1}{12}\sum_{i=1}^{k} n_i}}$$

3.2.1 Anisotropic Laplace trend

The main limitation of the Laplace trend is that it does not take into account the presence or the absence of activity in the system. To circumvent this problem, we replace the Laplace trend factor $U(k)$ with an anisotropic Laplace trend factor $A(k)$ that is defined as follows

$$A(k) = \begin{cases} g(U(k)) & \text{if no activity} \\ U(k) & \text{otherwise,} \end{cases}$$

where g is a "reliability growth-stopping" function as shown in Table 3.1. The g-function is chosen to satisfy $g(x) \to 0$ when $x \to \infty$ so that the reliability growth is stopped when there is no activity in the system.

The parameter σ of the Gaussian and Lorentzian g-functions may be estimated using tools from robust statistics as follows

$$\hat{\sigma} = 1.4826 \text{ MAD}\{(U(k) - U(k-1))_k\},$$

where MAD denotes the median absolute deviation [14].

Function	$g(x)$
Green [12]	$\frac{\tanh(x)}{2x}$ (if $x \neq 0$)
Gaussian [13]	$\exp\left(-\frac{x^2}{2\sigma^2}\right)$
Lorentzian [13]	$\frac{1}{1+x^2/\sigma^2}$

Table 3.1: reliability growth-stopping functions.

3.2.2 Weighted Laplace trend

Let $w \in [0,1]$ be a weight parameter denoting the proportion of software reliability growth during the period $[t_\ell, t_\ell + t_w]$. Then, we define a weighted failure intensity function as follows

$$\begin{aligned}\lambda_w(t) &= \lambda(t)\,\mathbf{1}_{(0\le t\le t_\ell)} + w\lambda(t)\,\mathbf{1}_{(t_\ell\le t\le t_\ell+t_w)} \\ &\quad + (1-w)\lambda(t_\ell)\,\mathbf{1}_{(t_\ell\le t\le t_\ell+t_w)},\end{aligned}$$

where $\lambda(t)$ is the failure intensity function, and $\mathbf{1}_S$ denotes the indicator function of a subset S.

When $w = 1$, the weighted failure intensity function reduces to the original intensity function, and when $w = 0$, the function λ_w becomes a constant (straight line). Moreover, for $w \in (0,1)$, the weighted failure intensity function λ_w has a less heavier tail that $\lambda(t)$ in the interval $[t_\ell, t_\ell + t_w]$ indicating a slow reliability growth of the Laplace trend as illustrated in Figure 3.1.

Therefore, we may define a weighted anisotropic Laplace test statistic as follows

$$\begin{aligned}A_w(k) &= A(k)\,\mathbf{1}_{(0\le k\le t_\ell)} + wA(k)\,\mathbf{1}_{(t_\ell\le k\le t_\ell+t_w)} \\ &\quad + (1-w)A(t_\ell)\,\mathbf{1}_{(t_\ell\le k\le t_\ell+t_w)},\end{aligned}$$

where $A(k)$ is the anisotropic Laplace test statistic.

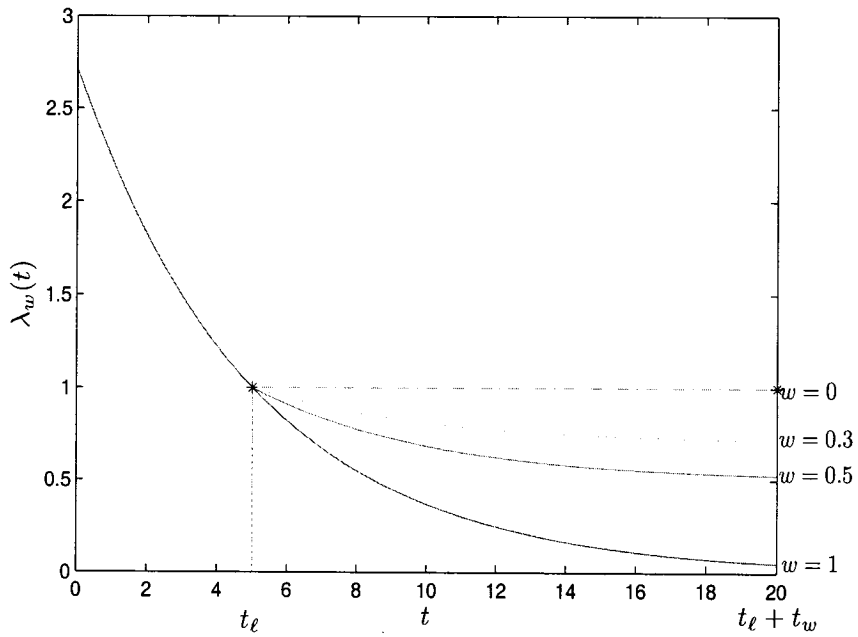


Figure 3.1: Weighted failure intensity function.

It can be shown that the 95% asymptotic efficiency on the standard Gaussian distribution is obtained with the tuning constant $\sigma = 2.3849$.

3.3 Experimental Results

We tested our proposed anisotropic Laplace test statistic on a real software failure data which was taken from an SAP development system. The data contains daily software failures that was recorded for a period of 175 days. Moreover, there are no activities in the system during the test phase process on the days 121, 122, 128, 142, 143, 144, 145, 146, 147, 148, 149, and 150.

Figure 3.2 displays the scatter plot of cumulative failure number versus failure time, and it clearly

illustrates an improving system since the probability of failures stabilizes substantially after a period of 150 days.

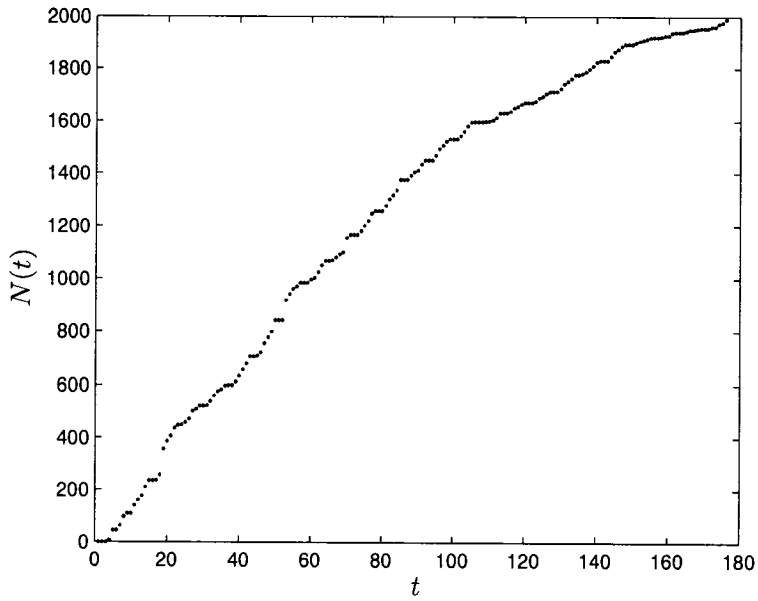


Figure 3.2: Cumulative Number of Failures vs. Failure Time.

3.3.1 Weighted Laplace trend results

Figure 3.3 through Figure 3.5 depict the much improved performance of the weighted anisotropic Laplace trends $A_w(k)$ with a weight parameter $w = 0.5$ in comparison with the anisotropic Laplace trends A_k. The no-activity periods of the weighted anisotropic Laplace trend are displayed with black-star points, whereas the no-activity periods of the anisotropic Laplace trend are displayed with red-star points.

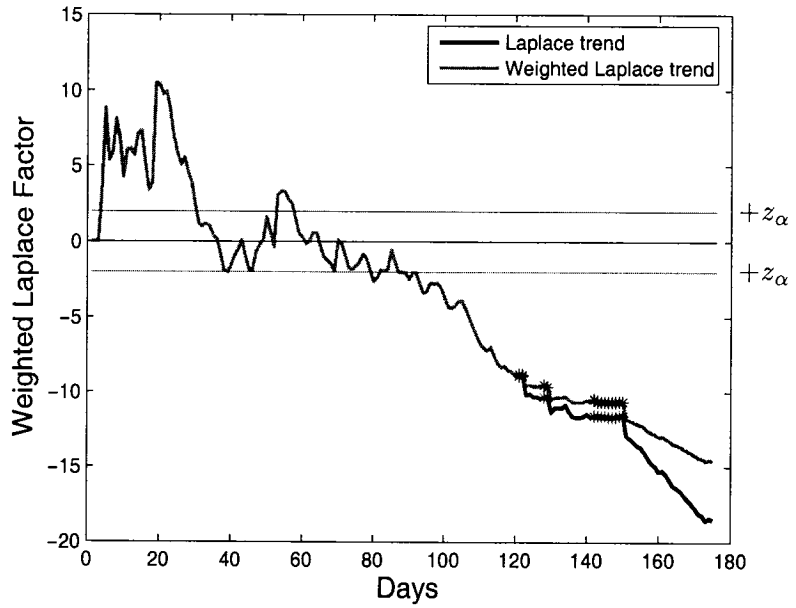


Figure 3.3: Weighted anisotropic Laplace trend using Green's function, with $w = 0.5$.

3.3.2 Weighted adjusted anisotropic Laplace trend results

The Laplace test statistic is a test for the null hypothesis H_0 that the data come from an HPP. Thus rejection of H_0 means that the process is not an HPP, but it could still in principle be a renewal process and hence still has no trend. In order to improve the test performance when the null hypothesis is a more general renewal process, the Lewis-Robinson (LR) test should be used [15]. The LR test is basically a scaled version of the Laplace test and it is defined as

$$U_{LR} = \frac{U(k)}{\widehat{CV}(\tau)},$$

where $\widehat{CV}(\tau)$ is an estimate of the coefficient of variation of the interfailure times τ_i, and it is calculated in terms of the mean and the standard deviation of interfailure times as follows

$$\widehat{CV}(\tau) = \frac{\sigma_\tau}{\bar{\tau}},$$

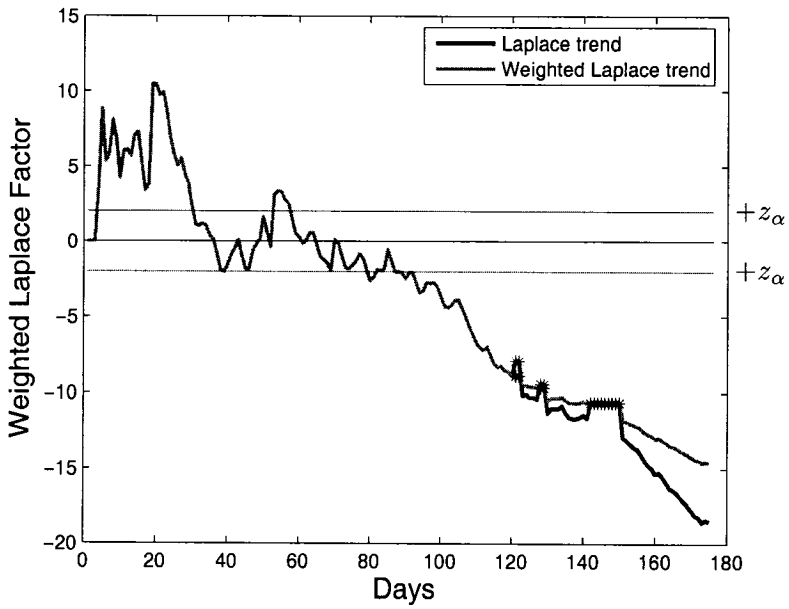


Figure 3.4: Weighted anisotropic Laplace trend using Gaussian function, with $w = 0.5$.

with τ representing the variable of interfailure times.

The reason for dividing the Laplace trend by the coefficient of variation is to account for non-exponential distributions of the interfailure times and also in order to insure that U_{LR} follows a standard normal distribution whenever the data come from a renewal process. Moreover, when the null-hypothesis is a renewal process model with non-exponential interarrival times, this adjustment maintains the type-I error probability better than the Laplace test [15].

Figure 3.6 through Figure 3.8 show the weighted adjusted anisotropic Laplace trends with a weight parameter $w = 0.5$. The weighted adjusted anisotropic Laplace trends with a weight parameter $w = 0.1$ are depicted in Figure 3.9 through Figure 3.11. Note that the no-activity periods of the weighted adjusted anisotropic Laplace trends are displayed with black-star points, whereas the no-activity periods of the adjusted anisotropic Laplace trends are displayed with red-star points.

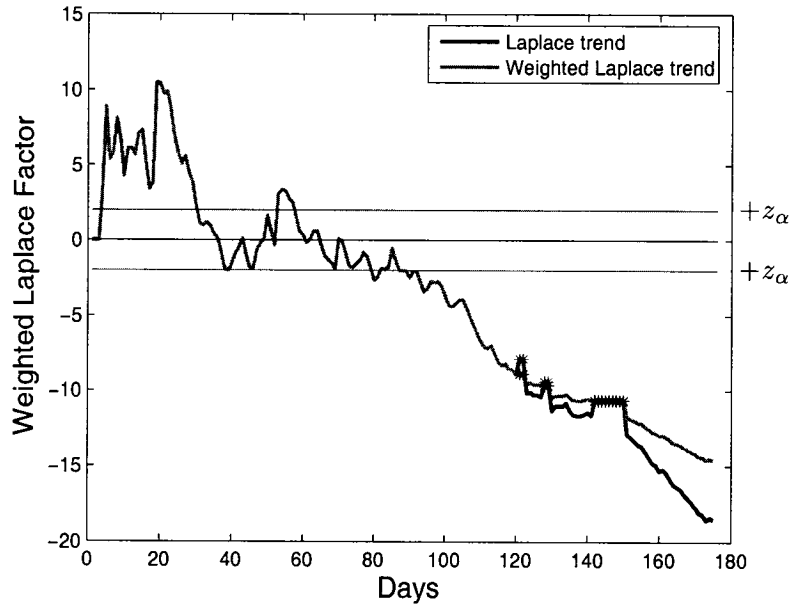


Figure 3.5: Weighted anisotropic Laplace trend using Lorentzian function, with $w = 0.5$.

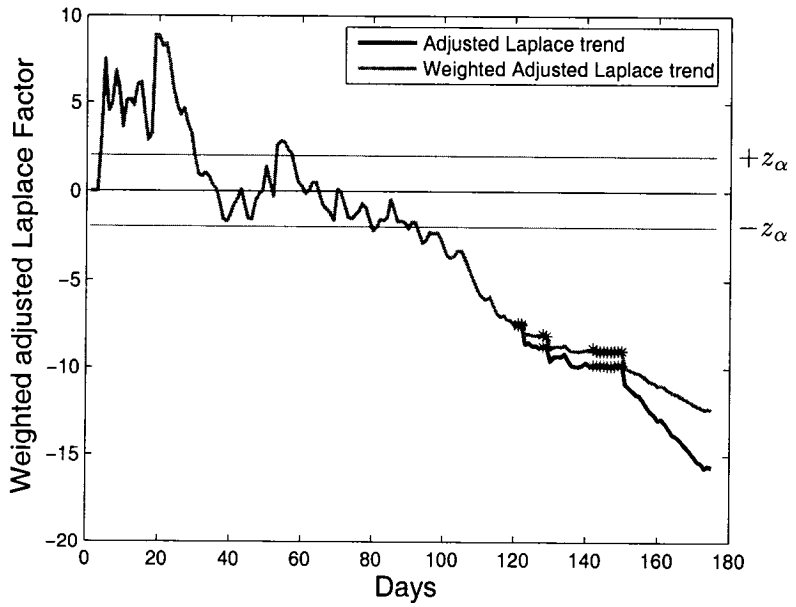


Figure 3.6: Weighted adjusted anisotropic Laplace trend using Green's function ($w = 0.5$).

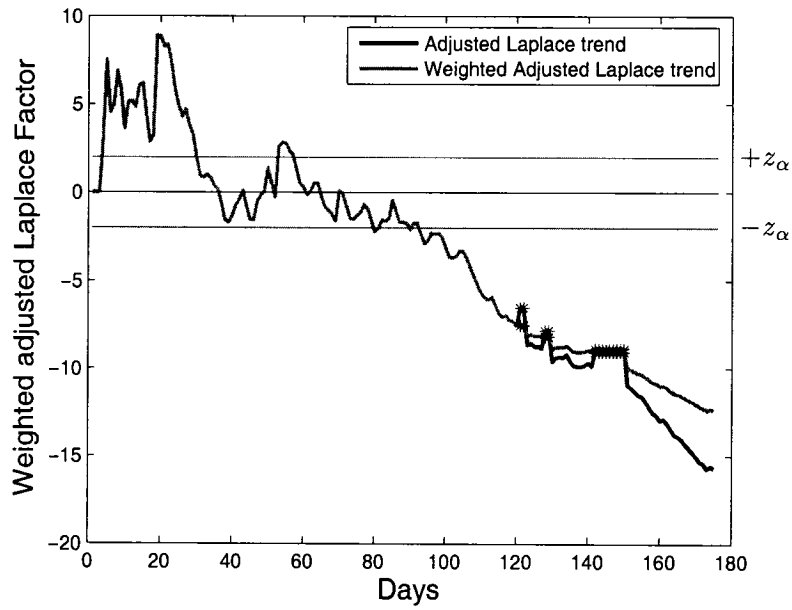


Figure 3.7: Weighted adjusted anisotropic Laplace trend using Gaussian function ($w = 0.5$).

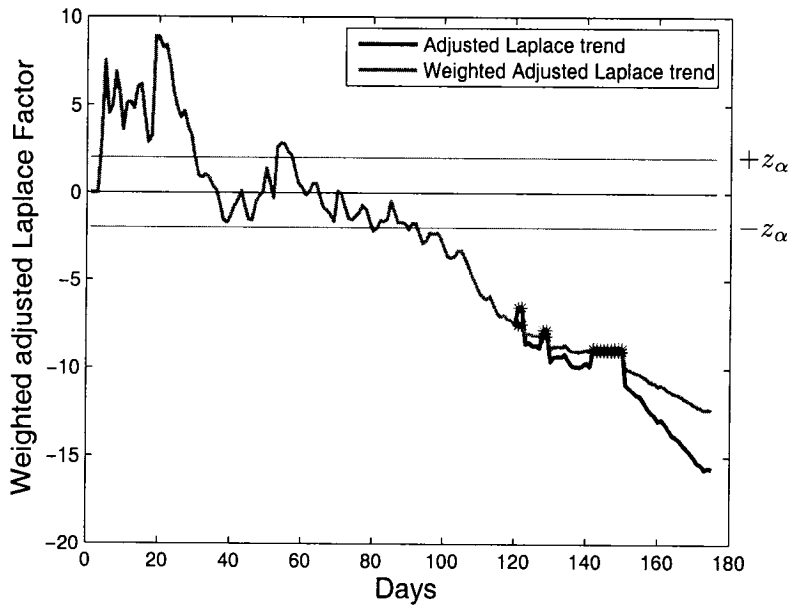


Figure 3.8: Weighted adjusted anisotropic Laplace trend using Lorentzian function ($w = 0.5$).

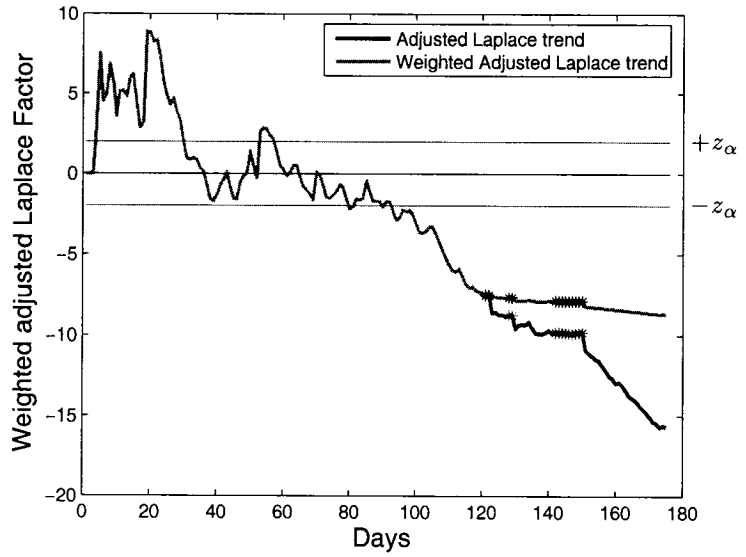


Figure 3.9: Weighted adjusted anisotropic Laplace trend using Green's function ($w = 0.1$).

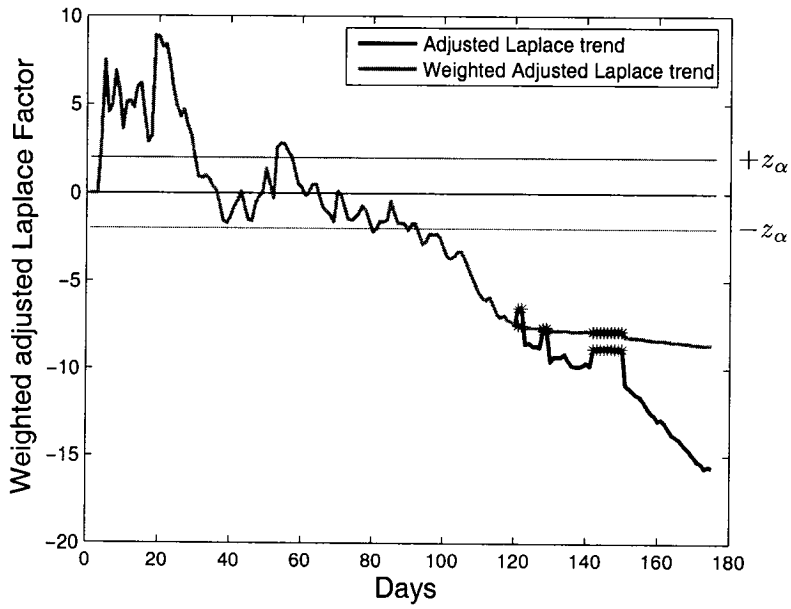


Figure 3.10: Weighted adjusted anisotropic Laplace trend using Gaussian function ($w = 0.1$).

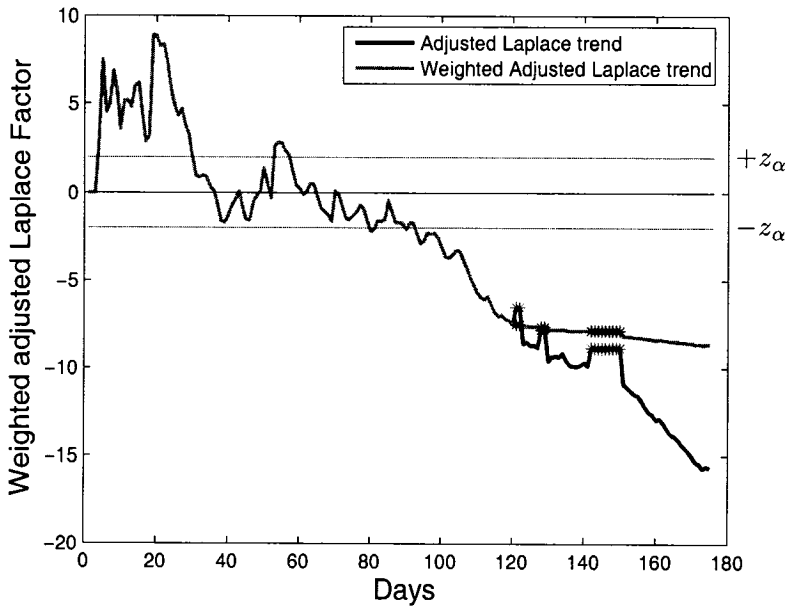


Figure 3.11: Weighted adjusted anisotropic Laplace trend using Lorentzian function ($w = 0.1$).

3.4 Conclusions

In this chapter, we proposed a new weighted Laplace test statistic for software reliability growth modelling. The proposed model not only takes into account the activity in the system but also the proportion of reliability growth within the model. This generalized approach is defined as a weighted combination of a growth reliability model and a non-growth reliability model. The experimental results clearly indicate a much improved performance of the proposed anisotropic Laplace test statistic in software reliability growth modelling.

CHAPTER 4

Operating Characteristic Curves-based Approach

We present a software defect prediction model using operating characteristic curves. The main idea behind our proposed technique is to use geometric insight in helping construct an efficient and fast prediction method to accurately predict the cumulative number of failures at any given stage during the software development process. Our predictive approach uses the number of detected faults instead of the software failure-occurrence time in the testing phase. Experimental results illustrate the effectiveness and the much improved performance of the proposed method in comparison with the Bayesian prediction approaches.

4.1 Introduction

Each software defect encountered by customers entails a significant cost penalty for software companies. Thus, knowledge about how many defects to expect in a software product at any given stage during its development process is a very valuable asset. Being able to estimate the number

of defects will substantially improve the decision processes about releasing a software product. Moreover, the production process for software products can be substantially improved by employing a prediction model that accounts for the dynamic nature of software production processes and reliably predicts the number of defects [2, 9, 16–18].

During the development process of computer software systems, many software defects may be introduced and often lead to critical problems and complicated breakdowns of computer systems [1]. Hence, there is an increasing demand for controlling the software development process in terms of quality and reliability. Software reliability can be evaluated by the number of detected faults. A software failure is defined as an unacceptable departure of program operation caused by a software fault remaining in the software system [2, 19]. In the traditional software development environment, software reliability evaluation, which shorten development intervals and reduce development costs, provides useful guidance in balancing reliability, time-to-market and development cost [9]. Hence, there is an increasing demand for prediction the quality and reliability of software.

Several software reliability prediction models have been proposed in the literature for estimating system reliability, but all these kinds of models make unrealistic assumptions to ensure solvability [4–8, 11, 15, 19]. These unreasonable assumptions in most traditional models limited the applications of these models [17, 18].

Bayesian statistics provide a framework for combining observed data with prior assumptions in order to model stochastic systems. Bayesian methods aim at assigning prior distributions to the parameters in the model in order to incorporate whatever *a priori* quantitative or qualitative knowledge we have available, and then to update these priors in the light of the data, yielding a posterior distribution via Bayes's Theorem [20]. The ability to include prior information in the model is not

only an attractive pragmatic feature of the Bayesian approach, but it is also theoretically vital for guaranteeing coherent inferences.

Motivated by the widely used concept of operating characteristic (OC) curves in statistical quality control to select the sample size at the outset of an experiment [21], we propose in this chapter a software defect prediction technique using OC curves in order to predict the cumulative number of failures at any given time. The core idea behind our proposed methodology is to use geometric insight in helping construct an efficient and fast prediction method to accurately predict the cumulative number of failures at any given time. Moreover, our predictive approach uses the number of detected faults instead of the software failure-occurrence time in the testing phase.

The layout of this chapter is organized as follows. In the next Section, a problem formulation is stated. In Section 3, we briefly review some Bayesian prediction models that will be used for comparison with our proposed approach. In Section 4, we propose a new prediction algorithm based on OC curves. In Section 5, we present experimental results to demonstrate the much improved performance of the proposed approach in the prediction of software defects. Finally, some conclusions are included in Section 6.

4.2 Problem Formulation

The problem addressed in this chapter may be concisely described by the following statement: Given the historical failure times data $\mathcal{D} = \{t_1, \ldots, t_n\}$ and its corresponding cumulative number of failures data $\mathcal{N} = \{N(t_1), \ldots, N(t_n)\}$, find the predicted cumulative number of failures at any given time t.

Model name	$m(t)$	$\lambda(t)$
Log-linear	$\frac{\exp(a+bt)}{b}$	$\exp(a+bt)$
Exponential	$a(1-\exp(-bt))$	$ab\exp(-bt)$
Power law	$\left(\frac{t}{a}\right)^b$	$\frac{b}{a}\left(\frac{t}{a}\right)^{b-1}$

Table 4.1: NHPP models.

4.3 Prediction using Bayesian Statistics

Assume we model the failure times using an NHPP with a parametrized failure intensity function $\lambda(t;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of unknown parameters. Table 4.1 shows examples of NHPP models with different failure intensity functions $\lambda(t;\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (a,b)$. Bayesian methods aim at assigning prior distributions to the parameters $\boldsymbol{\theta}$ is the model in order to incorporate whatever *a priori* quantitative or qualitative knowledge we have available, and then to update these priors in the light of the data, yielding a posterior distribution via Bayes's Theorem. The ability to include prior information in the model is not only an attractive pragmatic feature of the Bayesian approach, but it is also theoretically vital for guaranteeing coherent inferences.

4.3.1 Predictive density

Consider the problem of making prediction for a new failure time t without any measurements on the predictors for any of the individuals so that the dataset is just given by $\mathcal{D} = \{t_1, \ldots, t_n\}$. That is, we want to determine $p(t|\mathcal{D})$, the probability density function of the new failure time conditioned on the observed failure times. The function $p(t|\mathcal{D})$ is referred to as *predictive density*

of a new failure time and may be written in integral form as

$$p(t|\mathcal{D}) = \int p(t|\mathcal{D}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta},$$

where $p(\boldsymbol{\theta}|\mathcal{D})$ is the posterior distribution of $\boldsymbol{\theta}$ given by

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{D}) &= \frac{p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})} \\ &= \frac{p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}} \\ &= \frac{\{\prod_{i=1}^{n} p(t_i|\boldsymbol{\theta})\} p(\boldsymbol{\theta})}{\int \{\prod_{i=1}^{n} p(t_i|\boldsymbol{\theta})\} p(\boldsymbol{\theta}) d\boldsymbol{\theta}} \end{aligned}$$

and $p(\boldsymbol{\theta})$ is the prior distribution which represents information available about the unknown parameters. The prior estimate provides a means of combining exogenous information with observed data in order to estimate parameters of a probability distribution. It is convenient to choose simple forms of prior distributions which result in computationally tractable posterior distributions. Hence, the posterior distribution is found by combining the prior distribution $p(\boldsymbol{\theta})$ with the probability $p(\mathcal{D}|\boldsymbol{\theta})$ of observing the data given the parameters. The probability $p(\mathcal{D}|\boldsymbol{\theta})$ is also called the likelihood function of the data and it is given by

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(t_i|\boldsymbol{\theta}),$$

where

$$p(t_i|\boldsymbol{\theta}) = \lambda(t_i; \boldsymbol{\theta}) \exp\left(- \int_0^{t_i} \lambda(u; \boldsymbol{\theta}) du \right)$$

assuming that the failure times data are independent and identically distributed (iid). The likelihood function is the probability of observing the given data as a function of $\boldsymbol{\theta}$.

Therefore, the Bayesian approach consists of three main steps:

1. Assign prior distributions to all the unknown parameters.

2. Determine the likelihood of the data given the parameters.

3. Determine the posterior distribution of the parameters given the data.

Maximum Likelihood is a statistical estimator that can be used to estimate a models unknown parameter values from data. The maximum likelihood estimate (MLE) of $\boldsymbol{\theta}$ is that value of $\boldsymbol{\theta}$ that maximizes the likelihood function $p(\mathcal{D}|\boldsymbol{\theta})$ or equivalently that maximizes the log-likelihood function $\log(p(\mathcal{D}|\boldsymbol{\theta}))$, and it is the value that makes the observed data the most "probable".

4.3.2 Bayesian prediction

The Bayesian prediction approach proposed in [16] is based on the power law model shown in Table 4.1. The parameter b of the power law model may be estimated as follows

$$\widehat{b} = \frac{t_n}{\sum_{t=t_1}^{t_n} \log[N(t_n)/N(t)]},$$

and the predicted cumulative number of defects $N(t)$ at time t is given by

$$N(t) = N(t_n)\left(\frac{t}{t_n}F(2t, 2t_n; \gamma)\right)^{1/\hat{b}}, \tag{1}$$

where $\gamma = P\{\chi_n^2 \leq \chi_{\gamma,n}^2\}$, and $F(2t, 2t_n; \gamma)$ denotes the γ percentage point of the F-distribution with $2t$ and $2t_n$ degrees of freedom.

4.3.3 Bayesian prediction using MCMC

If we draw samples $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(N)}$ from the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$, then the predictive density may be approximated as follows

$$\begin{aligned} p(t|\mathcal{D}) &\approx \sum_{i=1}^{N} p(t|\mathcal{D}, \boldsymbol{\theta}^{(i)}) p(\boldsymbol{\theta}^{(i)}|\mathcal{D}) \\ &= \frac{1}{N} \sum_{i=1}^{N} p(t|\mathcal{D}, \boldsymbol{\theta}^{(i)}) \end{aligned}$$

The samples $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(N)}$ are draws from the posterior distribution of $\boldsymbol{\theta}$, and may be obtained using Markov chain Monte Carlo (MCMC) simulation algorithms [22, 23].

For the Bayesian prediction approach using MCMC, the predicted cumulative number of defects $N(t)$ at time t is also given by Eq. (1) where $\hat{b}$ is estimated using the MCMC algorithm [23].

Let $f(x|\theta)$ is Probability density function; $\pi(\theta)$ is the proposal density function; $g(\boldsymbol{\theta}^{(i)'}|\boldsymbol{\theta}^{(i)})$ is the proposal density function.

1. From Bayesian theorem,

$$\pi(\boldsymbol{\theta}^{(i)} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{\theta}^{(i)}) \pi(\boldsymbol{\theta}^{(i)})$$

2. Metropolis-Hastings

$$\eta = min\{1, \frac{\pi(\boldsymbol{\theta}^{(i)'} \mid \mathcal{D}) / g(\boldsymbol{\theta}^{(i)'}|\boldsymbol{\theta}^{(i)})}{\pi(\boldsymbol{\theta}^{(i)} \mid \mathcal{D}) / g(\boldsymbol{\theta}^{(i)}|\boldsymbol{\theta}^{(i)'})}\}$$

The algorithm of MCMC estimate parameters $\widehat{b}$ consists of the following steps:

1. using MCMC to simulate each parameter distribution.

2. estimate the maximal likely value of parameter distribution which gives us the value of expected parameter.

4.4 Proposed Method

Consider the two-sided hypothesis

$$H_0: \ t = t_k$$

$$H_1: \ t \neq t_k$$

where H_0 and H_1 are the null and the alternative hypotheses respectively.

Define $\chi^2_{\alpha,k}$ as the percentage value of the chi-square distribution with k degrees of freedom such that the probability that the chi-square distribution χ^2_n exceeds this value is α, that is

$$P\{\chi^2_k \geq \chi^2_{\alpha,k}\} = \alpha = P\{\text{reject } H_0 | H_0 \text{ is true}\},$$

where $\alpha \in (0, 1)$ is the probability of type I error (also referred to as the significance level).

Denote by

$$\beta = K\left(\chi^2_{\frac{\alpha}{2},\delta} - \frac{\delta}{\sigma\sqrt{n}}\right) - K\left(-\chi^2_{\frac{\alpha}{2},\delta} - \frac{\delta}{\sigma\sqrt{n}}\right),$$

where K is the cumulative distribution function of χ^2.

The predicted cumulative number of defects are given by:

$$N(T) = \frac{(\chi^2_{\beta,\alpha} + \chi^2_{\beta,\delta})^2\sigma^2}{\delta^2}$$

where: $\alpha = 0.1$, $\sigma = \sqrt{2n}$, $\delta = t_i, 1 = 1, 2, \ldots, n$, and $\chi^2_{\beta,\alpha}$ denotes the inverse of the chi-square cdf with α degrees of freedom at the values in β.

Suppose that H_0 is false and that the true value is $t = t_k + \delta$, where $\delta > 0$. Since H_1 is true, the distribution of the test statistic

$$Z = \frac{\chi^2_t - t_k}{\sqrt{2k}}$$

has a mean value equal to $\delta/\sqrt{2k}$, and a type II error will be made only if $-\chi^2_{\alpha/2} \leq Z \leq \chi^2_{\alpha/2}$. That is, the probability of type II error $\beta = P\{\text{accept } H_0 | H_0 \text{ is false}\}$ may be expressed as

$$\beta = \Phi\left(\chi^2_{\frac{\alpha}{2},t} - \frac{\delta}{\sqrt{2k}}\right) - \Phi\left(-\chi^2_{\frac{\alpha}{2},t} - \frac{\delta}{\sqrt{2k}}\right),$$

where Φ is the cumulative distribution function of χ^2_t.

The function $\beta(t)$ is evaluated by finding the probability that the test statistic Z falls in the acceptance region given a particular value of t. We define the operating characteristic (OC) curve of a test as the plot of $\beta(t)$ against t. Note that given the OC curve parameters β, α, k, and δ, we can derive the predicted cumulative number of defects at time t as follows

$$N(t) = \left(\frac{\sqrt{2k}}{\delta}\right)^2 \left(\chi^2_{\alpha,\delta} + \chi^2_{\beta,\delta}\right)^2. \tag{2}$$

Fig. 4.1 depicts a plot of the cumulative number of defects using OC curves.

The OC curve approach, however, makes a prediction without taking into account the historical data. To circumvent this limitation, we propose a predictive operating characteristic (POC) curve where the predicted cumulative number of defects at time t is calculated as follows

$$N(t) = \left(\frac{\sqrt{2p}}{\delta}\right)^2 \left(\chi^2_{\alpha,\delta} + \chi^2_{\beta,\delta}\right)^2, \tag{3}$$

and the parameter p is given by (see Fig. 4.2)

$$p = \begin{cases} N(t), & \text{if } t \leq t_n \\ N(t_n), & \text{if } t_n < t \leq T. \end{cases}$$

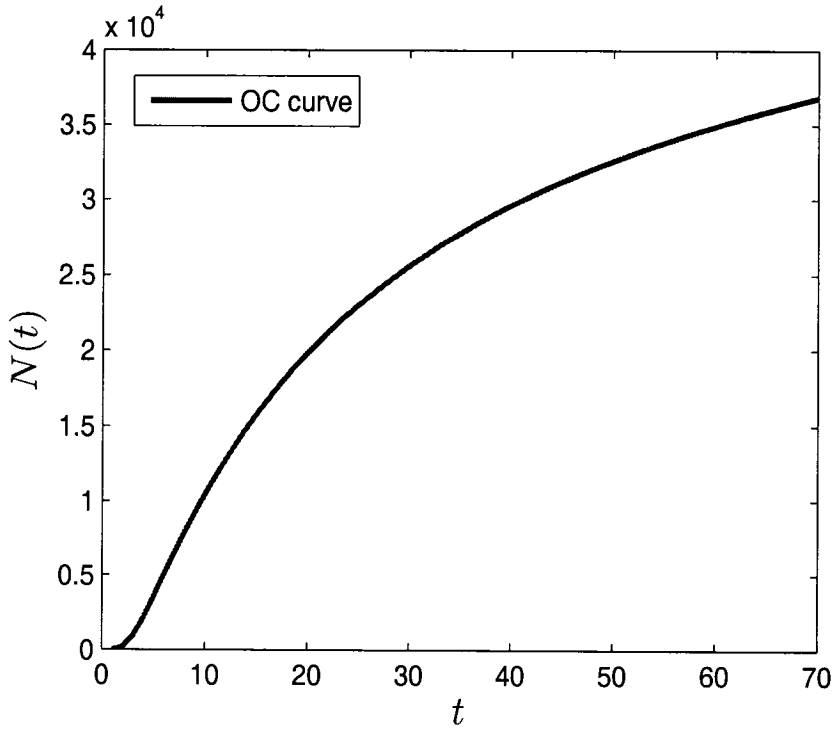


Figure 4.1: Illustration of cumulative number of defects using OC curves.

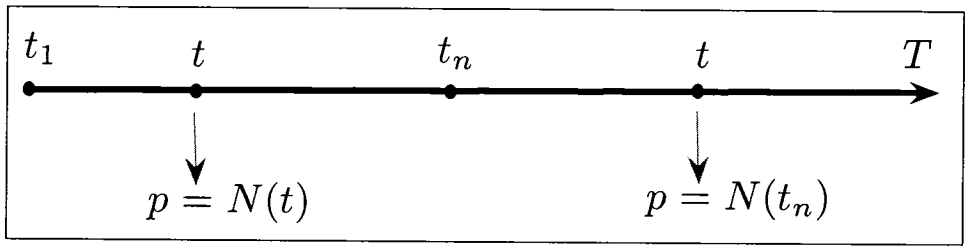


Figure 4.2: Illustration of the p parameter in the POC curve.

4.5 Experimental Results

We tested our proposed method on a real software failure dataset (DS I) that was taken from a SAP development system. This dataset contains monthly software failures that were recorded for a period of 60 months as shown in Table 4.2.

Fig. 4.3 depicts the cumulative number of failures versus failure time (month) during a software life cycle.

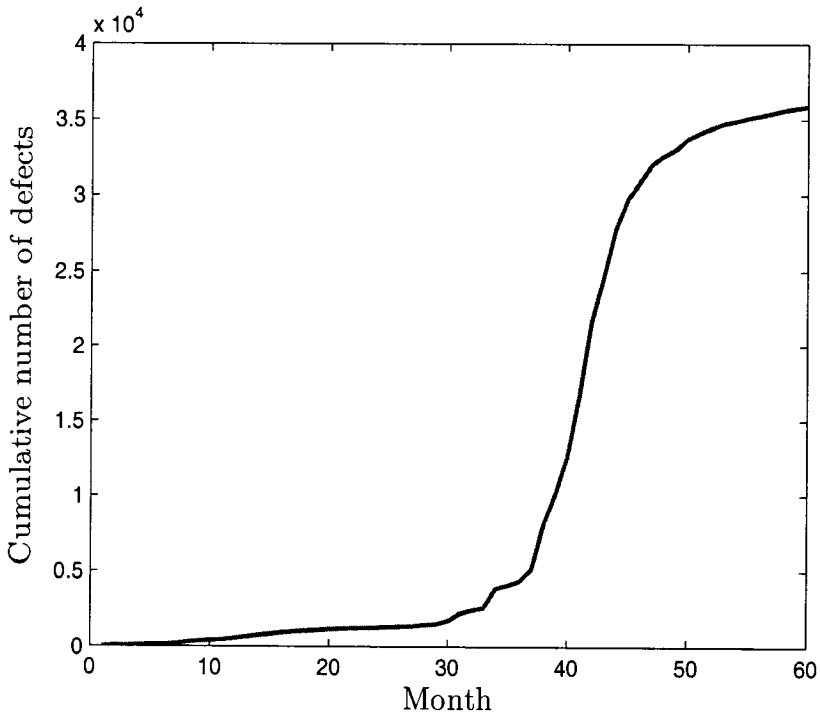


Figure 4.3: Cumulative Number of Failures vs. Failure Time (DS I)

We also applied the proposed method to a truncated dataset (DS II) that was obtained by truncating the original software failure data after the 40th month as shown in Fig. 4.4. Note that the cumulative number of failures stabilizes substantially after the 50th month, which clearly indicate

Month	Cumulative number of defects	Month	Cumulative number defects
1	17	31	2,217
2	39	32	2,430
3	53	33	2,586
4	87	34	3,884
5	106	35	4,099
6	140	36	4,385
7	165	37	5,104
8	286	38	8,074
9	359	39	10,120
10	412	40	12,618
11	461	41	16,715
12	555	42	21,606
13	654	43	24,592
14	747	44	27,789
15	836	45	29,739
16	926	46	30,843
17	989	47	32,011
18	1,049	48	32,599
19	1,103	49	33,010
20	1,152	50	33,707
21	1,182	51	34,103
22	1,213	52	34,426
23	1,225	53	34,736
24	1,266	54	34,903
25	1,306	55	35,110
26	1,331	56	35,261
27	1,363	57	35,440
28	1,443	58	35,614
29	1,495	59	35,763
30	1,737	60	35,876

Table 4.2: Software failure data.

that the system is improving.

In all the experiments, we used a probability of type I error $\alpha = 0.01$. The value of γ was set to $1 - \alpha$.

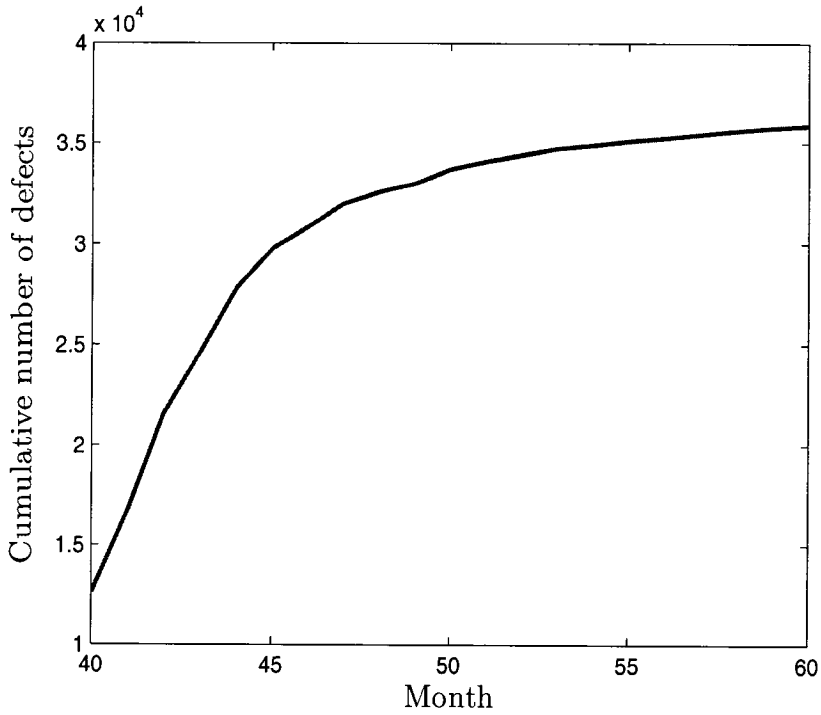


Figure 4.4: Cumulative Number of Failures vs. Failure Time (DS II)

4.5.1 Qualitative evaluation of the POC method

In this subsection, we present simulation results where the Bayesian prerdiction method [16], the Bayesian prediction using MCMC [23], OC curve approach, and the POC curve algorithm are applied to the software failure dataset (DS I) and also to the truncated software failure data (DS II).

For the Bayesian prediction method, the estimate of the parameter $\widehat{b}$ is equal to 0.3374, and for

the Bayesian prediction approach with MCMC the estimate of the value of $\widehat{b}$ is equal to 0.5402.

Fig. 4.5 and Fig. 4.6 show the prediction results of the proposed POC curve in comparison the Bayesian approaches for both datasets DS I and DS II respectively. These results clearly indicate that our method outperforms the Bayesian techniques used for comparison. Moreover, the proposed method is simple and easy to implement. One main advantage of the proposed algorithm is the nearly perfect fit between the predicted data and the observed data.

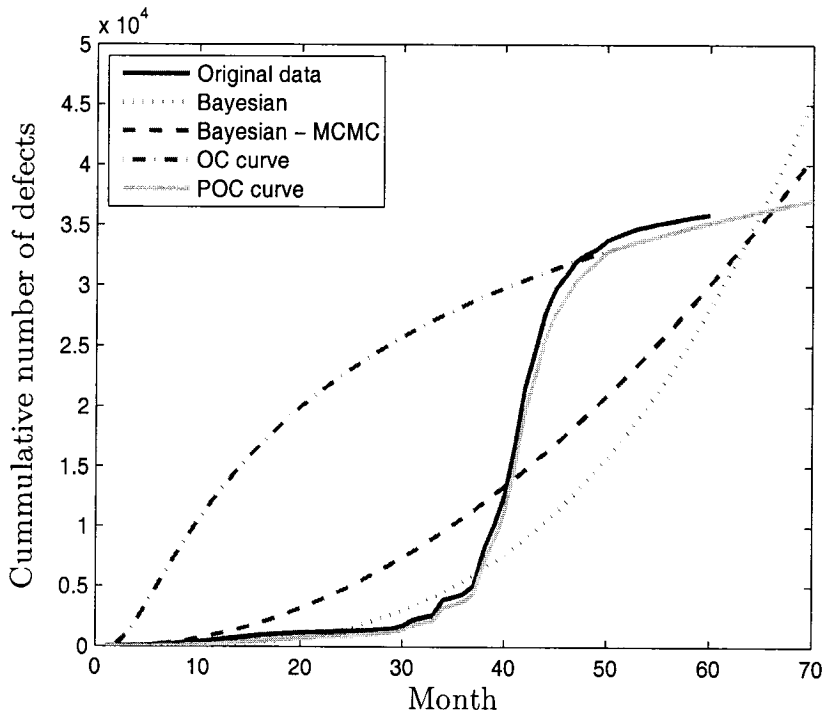


Figure 4.5: Comparison of the prediction results for DS I.

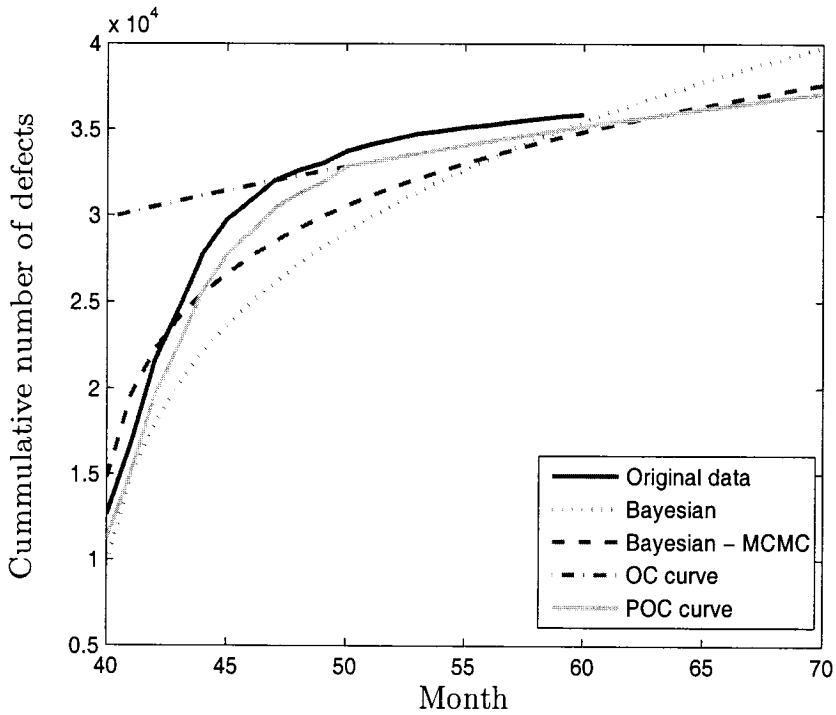


Figure 4.6: Comparison of the prediction results for DS II.

4.5.2 Quantitative evaluation of the POC method

Denote by $N_o(t)$ and $N_p(t)$ the observed and the predictive cumulative number of failures respectively.

To quantify the better performance of the proposed predictive method in comparison with the Bayesian approaches, we computed three goodness-of-fit measures: the skill score, the Nash-Sutcliffe model efficiency coefficient, and the relative error between the observed $T_o \times 2$ data matrix

$$\mathcal{D}_o = \{(t, N_o(t)) : t = 1, \ldots T_o\},$$

and the predicted $T_p \times 2$ data matrix

$$\mathcal{D}_p = \{(t, N_p(t)) : t = 1, \ldots T_p\}.$$

Note that the size of observed data matrix $\mathcal{D}_o$ may not be equal to the size of the predicted data matrix $\mathcal{D}_p$, and hence an intersection step is necessary to pair up the observed data to the predicted data. This intersection function is setup to pair up the first column in the observed data matrix and the first column in the predicted data matrix. Data values are located in the second column of both matrices. More precisely, we create a subset of matched data $\mathcal{D}_m = \{t, N_o(t), N_o(t) : t = 1, \ldots T_m\}$ that would be used to compute the following goodness-of-fit measures:

1. **Skill Score**: it is a error statistic that is used to quantify the accuracy of prediction models, and it defined as follows

$$SS = 1 - \frac{\sqrt{\frac{1}{T_m}\sum_{t=1}^{T_m}\left(N_o(t) - N_p(t)\right)^2}}{\sqrt{\frac{1}{T_m - 1}\sum_{t=1}^{T_m}\left(N_o(t) - \overline{N}_o\right)^2}},$$

   The model prediction is better, when the value of the skill score SS is closer to one. When SS is less than zero, the model predictions are poor and the model errors are greater than observed data variability.

2. **Nash-Sutcliffe model efficiency coefficient**: is an indicator of the model's ability to predict about the 1:1 line between the observed and the predicted data, and it is defined as follows

$$E = 1 - \frac{\sum_{t=1}^{T_m}\left(N_o(t) - N_p(t)\right)^2}{\sum_{t=1}^{T_m}\left(N_o(t) - \overline{N}_o\right)^2}.$$

   The Nash-Sutcliffe model efficiency coefficient is a statistic similar to the skill score in that the closer to one the better the model prediction. A value of $E = 1$ indicates that the model

prediction is perfect, and if the value of E is equal to or less than zero, then the model prediction is considered poor.

3. **Relative error**: it measures how close a model is estimated with respect to the actual data. The relative error (RE) is defined as

$$RE = \frac{N_p(t) - N_o(t)}{N_o(t)}, \quad t = 1, \ldots, T_m$$

The values of the three goodness-of-fit measures for all the experiments are depicted in Fig. 4.7 through Fig. Fig. 4.12, which clearly show that the proposed method gives the best results indicating the consistency with the subjective comparison.

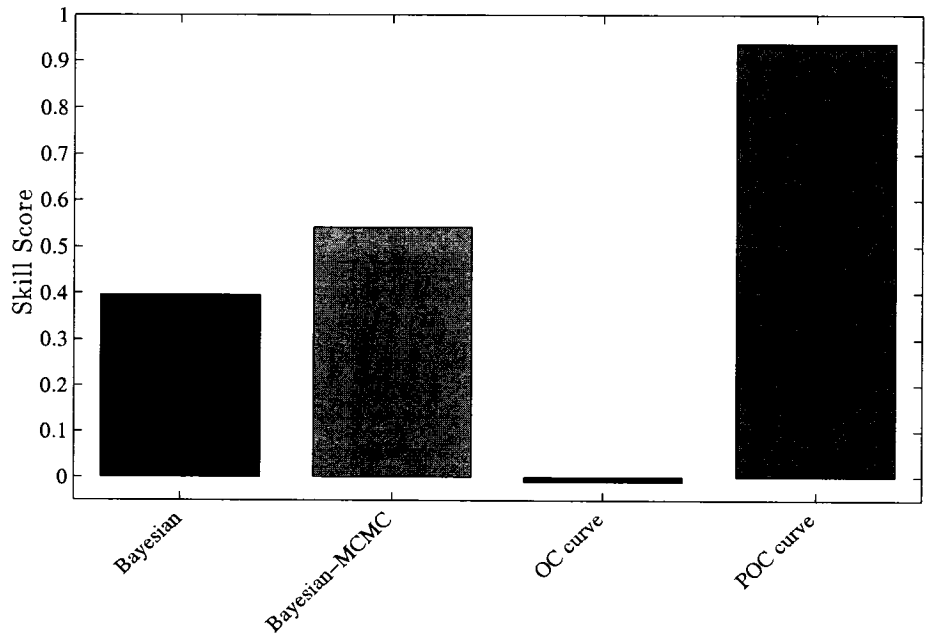


Figure 4.7: Skill score results for DS I.

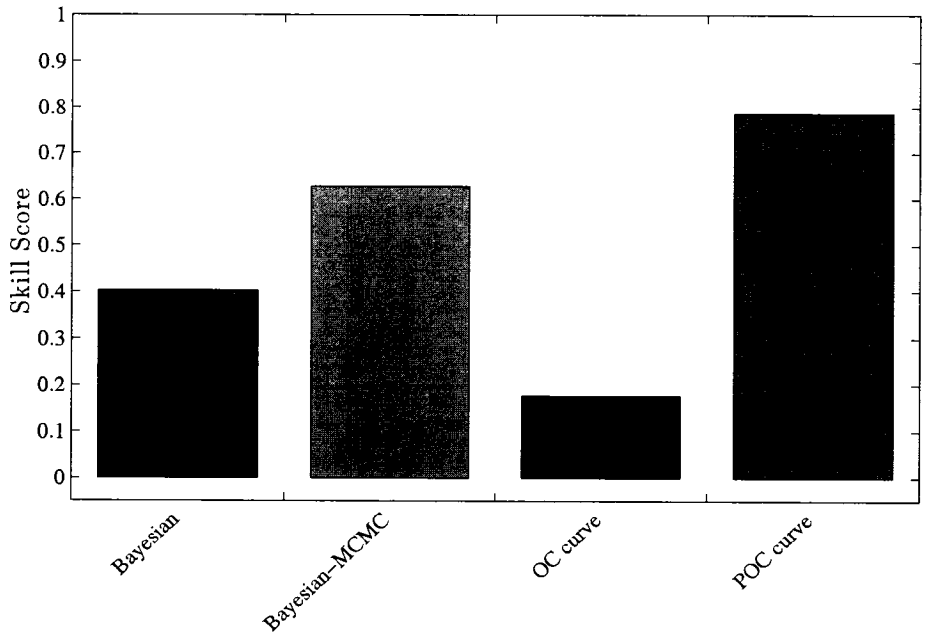


Figure 4.8: Skill score results for DS II.

4.6 Conclusions

In this chapter, we introduced a new method for software defects prediction using operating characteristic curves. The core idea behind our proposed technique is to reliably predict the cumulative number of defects at any given stage during the software development process. The prediction accuracy of the proposed approach is validated on a real software failure data using several goodness-of-fit measures. The experimental results clearly show a much improved performance of the proposed approach in comparison with the Bayesian prediction methods.

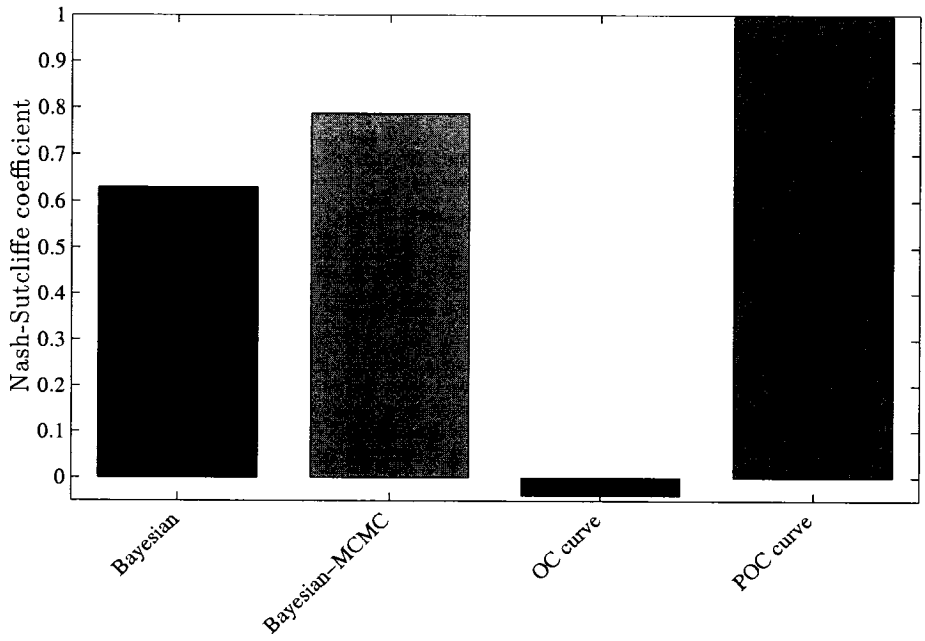


Figure 4.9: Nash-Sutcliffe model efficiency coefficient results for DS I.

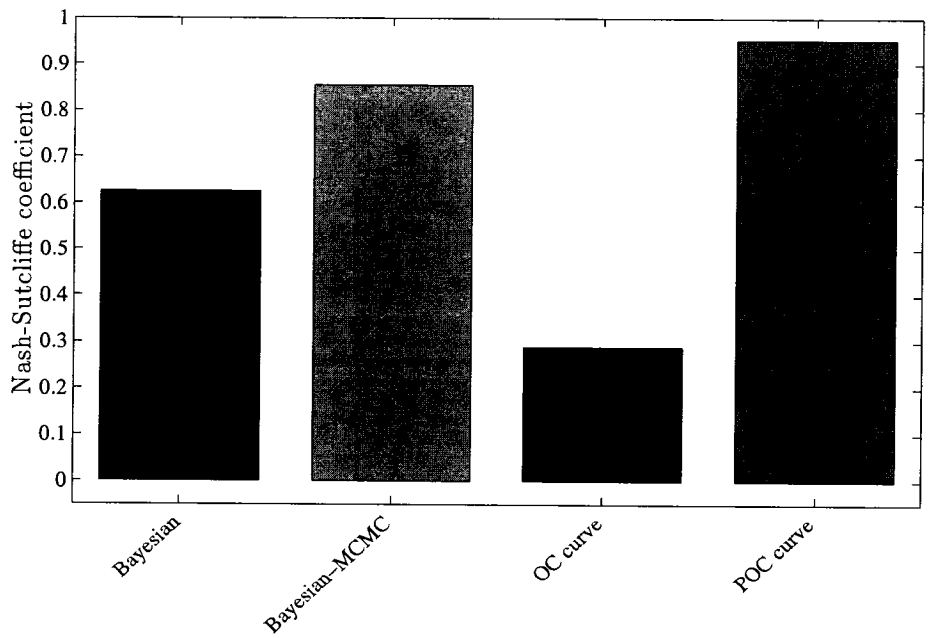


Figure 4.10: Nash-Sutcliffe model efficiency coefficient results for DS II.

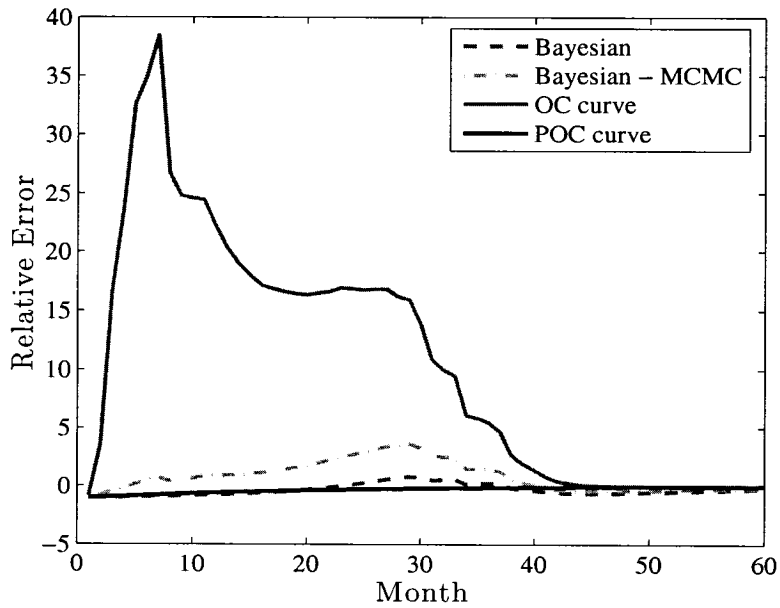


Figure 4.11: Relative error results for DS I.

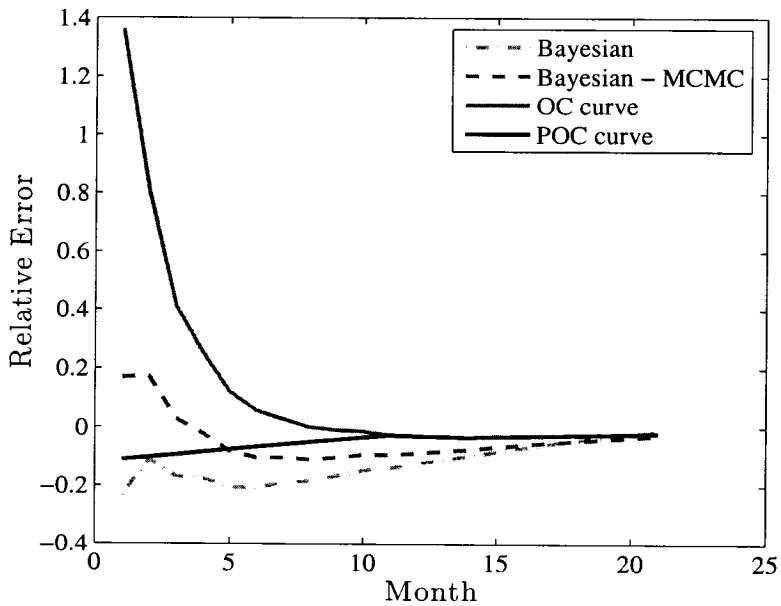


Figure 4.12: Relative error results for DS II.

CHAPTER 5

Conclusions and Future Work

A defect prediction solution provides a guideline to the sources of defects that might be caused due to programmers inability, failure in requirements collection or design mistakes. Thus, a defect prediction model with source identification can give important ideas regarding the erroneous bottlenecks in the software development cycle. Especially, efficiency focused software development units can benefit using defect cause information. They can take necessary precautions in a proactive manner. In other words, a defect focused prediction solution can also help to change the development methods. Such a solution or systematic approach can affect in a positive manner to produce less defected software.

An important aspect of a defect prediction solution is that such a solution becomes necessary when there is a trade-off between to deliver earlier and to deliver with fewer defects. In today's software development industry, all companies and software development houses are in a severe competition that minimizing development time decreases the overall project cost [27, 28]. On the other hand, less development and testing time also increases the defect density ratio in the final product. So, with this fact the executive management of the software company should require a quantitative indicator to find the correct point in this balance. Therefore a defect prediction solution

may provide the required quantitative metric to make a decision on the product delivery. The senior management of the software development company would be able to decide launching the product if the defect density level is below a certain threshold.

This thesis has presented statistical defect prediction models for software quality assurance. We have demonstrated the performance of the proposed algorithms through a variety of software failure datasets, and we compared our techniques with existing defect prediction methods.

In the next Section, the contributions made in each of the previous chapters and the concluding results drawn from the associated research work are presented. Suggestions for future research directions related to this thesis are provided in Section 5.2.

5.1 Contributions of the Thesis

5.1.1 Anisotropic Laplace trend

We proposed a nonlinear measure of trend for software reliability growth enhancement. The proposed test statistic is defined in terms of redescending stabilization functions which tackle the problem of increasing reliability growth by taking into account the activity in the system. The experimental results clearly indicate a much improved performance of the proposed anisotropic Laplace test statistic in software reliability growth enhancement.

5.1.2 Weighted anisotropic Laplace test statistic

We proposed a new weighted Laplace test statistic for software reliability growth modelling. The proposed model not only takes into account the activity in the system but also the proportion of

reliability growth within the model. This generalized approach is defined as a weighted combination of a growth reliability model and a non-growth reliability model. The experimental results clearly indicate a much improved performance of the proposed anisotropic Laplace test statistic in software reliability growth modelling.

5.1.3 Operating characteristic curves-based approach

We introduced a new method for software defects prediction using operating characteristic curves. The core idea behind our proposed technique is to reliably predict the cumulative number of defects at any given stage during the software development process. The prediction accuracy of the proposed approach is validated on a real software failure data using several goodness-of-fit measures. The experimental results clearly show a much improved performance of the proposed approach in comparison with the Bayesian prediction methods.

5.2 Future Research Directions

Several interesting research directions motivated by this thesis are discussed next. In addition to designing robust statistical models for software defect prediction, we intend to accomplish the following projects in the near future:

5.2.1 Predictive operating characteristic curves and Laplace trend

The POC curve method introduced in Chapter 4 provides a reliable software defect prediction technique. The performance of this approach, however, is not very satisfactory when the Laplace trend statistic continues to increase, that is when the software reliability deteriorate rapidly indicating a

highly increasing failure intensity.

To circumvent this limitation, we combine the POC curve approach with Laplace trend analysis to improve the prediction performance. The key idea is to calculate the "Laplace trend stopping increase" point $t = t_s$ as shown in Figure 5.1. Then, we use the POC curve method when the Laplace trend statistic starts to decrease, that is when $t \in (t_s, ..., T)$.

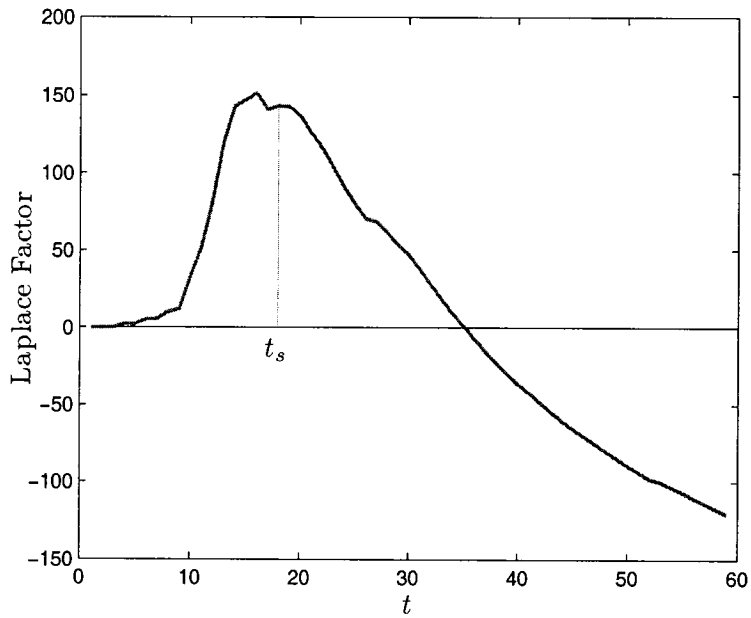
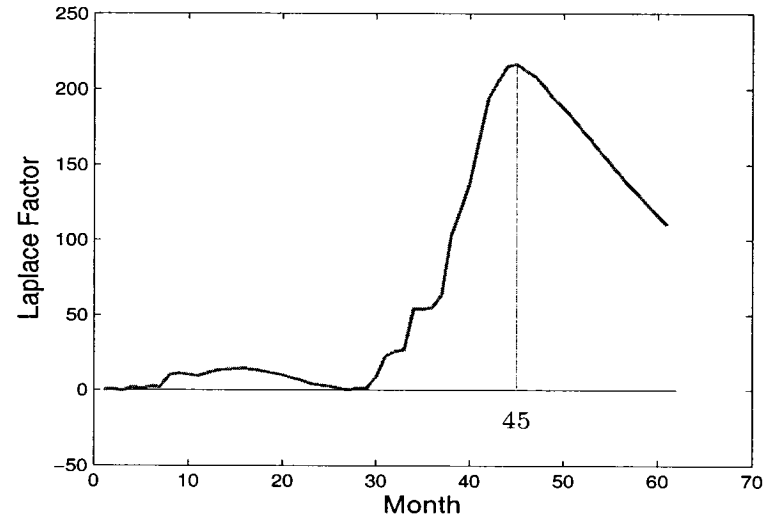
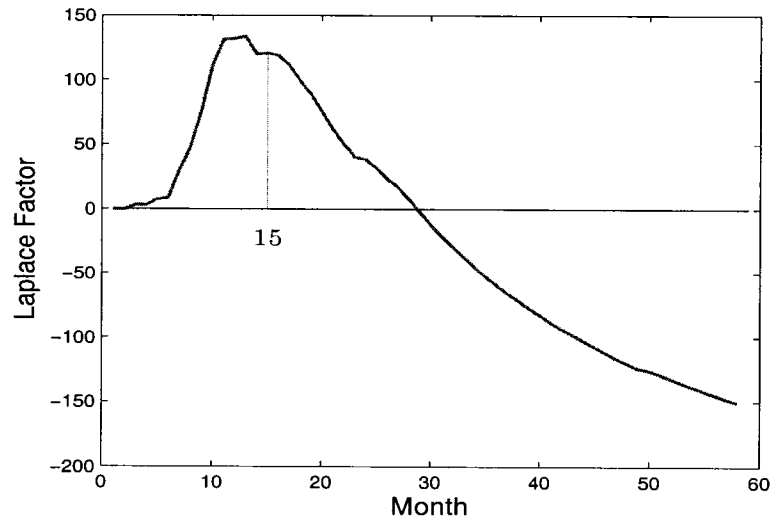


Figure 5.1: Laplace Factor vs. Failure Time.

Our preliminary results shown in Fig. 5.2(a) and Fig. 5.2(b) clearly illustrate that after the 45^{th} month for DS I and after the 15^{th} month for DS II, the Laplace trend statistic starts to decrease.



(a)



(b)

Figure 5.2: Laplace Factor vs. Failure Time: (a) DS I, (b) DS II.

5.2.2 Recurrent neural networks

Artificial neural network (ANN) models have been extensively studied with the aim of achieving human-like performance, especially in the field of pattern recognition. These networks are composed of a number of nonlinear computational elements which operate in parallel and are arranged in a manner reminiscent of biological neural interconnections. The area of neural networks is nowadays considered from two main perspectives. The first perspective is cognitive science and the second perspective is information processing. The neural networks in this thesis are approached from an engineering perspective, i.e. to make networks efficient in terms of topology, learning algorithms, ability to approximate functions and capture dynamics of time-varying systems.

The classic approach to time series prediction is to undertake an analysis of the time series data, which includes modeling, identification of the model and model parameter estimation phases. The design may be iterated by measuring the closeness of the model to the real data. This can be a long process, often involving the derivation, implementation and refinement of a number of models before one with appropriate characteristics is found. In particular, the most difficult systems to predict are:

- those with non-stationary dynamics, where the underlying behavior varies with time.
- those which deal with physical data which are subject to noise and experimentation error.
- those which deal with short time series, providing few data points on which to conduct the analysis.

A recurrent neural network is a neural network where the connections between the units form a directed cycle. Recurrent neural networks must be approached differently from feedforward neural networks, both when analyzing their behavior and training them. Recurrent neural networks can

also behave chaotically. Usually, dynamical systems theory is used to model and analyze them. A popular type of recurrent neural network is the Elman network.

5.2.3 Machine learning approaches

As a broad subfield of artificial intelligence (AI), machine learning is concerned with the design and development of algorithms and techniques that allow computers to "learn". As regards machines, one might say, very broadly, that a machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves. The major focus of Machine learning research is to extract information from data automatically by computational and statistical methods, hence, machine learning is closely related to data mining and statistics but also theoretical computer science.

Machine learning usually refers to the changes in systems that perform tasks associated with AI. Such tasks involve recognition, diagnosis, planning, prediction, etc. The "changes" might be either enhancements to already performing systems or synthesis of new systems.

One might ask "Why should machines have to learn? Why not design machines to perform as desired in the first place?" There are several reasons why machine learning is important. Some of these are:

- Some tasks cannot be defined well except by example, that is, we might be able to specify input/output pairs but not a concise relationship between inputs and desired outputs. We would like machines to be able to adjust internal structure to produce correct outputs for a large number of sample inputs and thus suitably constrain their input/output function to approximate the relationship implicit in the examples.
- It is possible that hidden among large piles of data are important relationships and

correlations. Machine learning methods can often be used to extract these relationships (data mining).

- Human designers often produce machines that do not work as well as desired in the environments in which they are used. In fact, certain characteristics of the working environment might not be completely known at design time. Machine learning methods can be used for on-the-job improvement of existing machines designs.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it that humans would want to write down.
- Environments change over time. Machines can adopt to a changing environment would reduce the need for constant redesign.
- New knowledge about tasks is constantly being discovered by humans. Continuing redesign of AI systems to conform to new knowledge is impractical, but machine learning methods might be able to track much of it.

There are two major settings in which we wish to learn a function f: *supervised* and *unsupervised*. In supervised learning, we know the values of f for the m samples in the training set $\mathbb{S}$. We assume that if we can find a hypothesis h that closely agrees with f for the members of $\mathbb{S}$, then this hypothesis will be a good guess for f, especially if $\mathbb{S}$ is large. Curve fitting is a simple example of supervised learning of a function.

In unsupervised learning, we simply have a training set of vectors without function values of them. The problem in this case, typically, is to partition the training set into subsets $\mathbb{S}_1, \ldots, \mathbb{S}_k$ in some appropriate way.

Our future efforts will be focused on evaluating various machine learning models to develop

robust prediction approaches. The performance of each prediction method will be evaluated regarding their precision, recall, robustness and sensitivity using confusion matrices and simulations. A model's precision is defined as the ratio of the number of modules correctly predicted as defective, or true positive (t_p), to the total number of modules predicted as defective in the set ($t_p + f_p$). A model's recall is defined as the ratio of the number of modules predicted correctly as defective (t_p) to the total number of defective modules in the set ($t_p + f_n$). To perform well, a model must achieve both high precision and high recall. However, a trade-off exists between precision and recall. For example, if a model predicts all modules as defective, its recall will be equal to 1, but the precision will be low. Obviously, in this case, we can't say that the model performs well. On the other hand, if a model predicts only one module as defective and the prediction turns out to be correct, the model's precision will be equal to 1. Yet again, we cannot consider this model to have good performance. In this project, we propose the use of the F-measure to measure prediction performance. F-measure considers precision and recall equally important by taking their harmonic mean. Like precision and recall, F-measure is always valued between 0 and 1, and a higher value indicates better prediction performance.

List of References

[1] J.D. Musa, "A theory of software reliability and its application," *IEEE Transactions on Software Engineering*, vol. 1, no. 1, pp. 312-327, 1975.

[2] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Company, 1987.

[3] A.L. Goel and K. Okumoto, "Time-dependent error detection rate models for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206-211, 1979.

[4] M.R. Bastos Martini, K. Kanoun, and J. Moreira de Souza, "Software-reliability evaluation of the TROPICO-R switching system," *IEEE Transactions on Reliability*, vol. 39, no. 3, pp. 369-379, 1990.

[5] K. Kanoun and J.C. Laprie, "Software reliability trend analysis from theoretical to practical considerations," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 525-532, 1992.

References

[6] A.L. Goel, "Software reliability models: assumptions, limitations and applicability," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1411-1423, 1985.

[7] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, vol. 32, no. 5, pp. 475-485, 1983.

[8] J.H. Lo and C.Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," *The Journal of Systems and Software*, vol. 79, pp. 1312-1323, 2006.

[9] X. Zhang and H. Pham, "Software field failure rate prediction before software deployment," *The Journal of Systems and Software*, vol. 79, pp. 291-300, 2006.

[10] D.R. Cox and P.A.W. Lewis, *The Statistical Analysis of a Series of Events*, Chapman and Hall, London, 1978.

[11] O. Gauodin, "Optimal properties of the Laplace trend test for software-reliability models," *IEEE Transactions on Reliability*, vol. 20, no. 9, pp. 740-747, 1992.

[12] P.J. Green, "Bayesian Reconstruction from Emission Tomography Data Using Modified EM Algorithm," *IEEE Transactions on Medical Imaging*, vol. 9, no. 1, pp. 84-93, 1990.

[13] P. Perona and J. Malik, "Scale space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine intelligence*, vol. 12, no. 7, pp. 629-639, 1990.

[14] P.J. Rousseeuw and A.M. Leroy, *Robust Regression and Outlier Detection*, Wiley, NY, 1987.

[15] H.E. Ascher, C.K.Hansen, "Spurious exponentiality observed when incorrectly fitting a distribution to nonstationary data," *IEEE Transactions on Reliability*, vol. 47, no. 4, pp. 451-45, 1998.

[16] J.W. Yu, G.L. Tian, and M.L. Tang, "Predictive analyses for nonhomogeneous Poisson processes with power law using Bayesian approach," *Computational Statistics & Data Analysis*, 2007.

[17] C.G. Bai, "Bayesian network based software reliability prediction with an operational profile," *Journal of Systems and Software*, vol. 77, no. 2, pp. 103-112, 2004.

[18] N.E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol.5, no. 5, pp. 675-689, 1999.

[19] A.L. Goel and K. Okumoto, "Time-dependent error detection rate models for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206-211, 1979.

[20] W.M. Bolstad, *Introduction to Bayesian Statistics*, John Wiley, 2004.

[21] D.C. Montgomery, *Introduction to Statistical Quality Control*, John Wiley & Sons, 2005.

[22] W.R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov chain Monte Carlo in Practice*, Chapman & Hall/CRC, 1995.

[23] C. Robert, *Bayesian Choice*, 2nd Edition, Springer Verlag, NY, 2001.

[24] M.R. Lyu, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill, 1996.

[25] ANSI/IEEE, Standard Glossary of Software Engineering Terminology, STD-729-1991, *ANSI/IEEE*, 1991.

[26] B. Littlewood and L. Strigini, "Software Reliability and Dependability: A Roadmap," *Proc. 22nd International Conference on Software Engineering*, Limerick, pp. 177-188, 2000.

[27] P.C. Pendharkar, G.H. Subramanian, and J. Rodger, "A probabilistic model for predicting software development effort," *IEEE Transactions on Software Engineering*, vol. 31, no.7, pp. 615-624, 2005.

[28] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," *Proc. ACM ICSE conference*, 2005.

[29] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no.12, pp. 736-743, 1997.

[30] F. Lanubile and G. Visaggio, "Evaluating predictive quality models derived from software measures: lessons learned," *Journal of Systems and Software*, vol.38, pp. 225-234, 1997.

[31] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer, 1996.

[32] D. Heckerman, "Bayesian Networks for Data Mining," *Data Mining and Knowledge Discovery*, vol. 1, pp. 79-119, 1997.

[33] D. Partridge, *Artificial Intelligence and Software Engineering*, AMACOM, 1998.

[34] R. S. Michalski, I. Bratko and M. Kubat (ed.), *Machine Learning and Data Mining: Methods and Applications*, John Wiley & Sons Ltd., 1998.

[35] K. Shukla, "Neuro-genetic prediction of software development effort," *Information and Software Technology*, vol.42, no.10, pp. 701-713, 2000.

[36] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 126-137, 1995.

[37] G. Finnie, G. Wittig, and J-M. Desharnais, "A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models," *Journal of Systems and Software*, vol.39, no.3, pp. 281-289, 1997.

[38] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675-689, 1999.

[39] T. Dohi, Y. Nishio, and S. Osaki, "Optimal software release scheduling based on artificial neural networks," *Annals of Software Engineering*, vol.8, no.1, pp. 167-185, 1999.

[40] W. Cohen and P. Devanbu, "A comparative study of inductive logic programming for software fault prediction," *Proc. the 14th International Conference on Machine Learning*, 1997.

[41] M. de Almeida, H. Lounis and W. Melo, "An investigation on the use of machine learned models for estimating correction costs," *Proc. International Conference on Software Engineering*, pp.473-476, 1998.

[42] M. de Almeida and S. Matwin, "Machine learning method for software quality model building," *Proc. International Symposium on Methodologies for Intelligent Systems*, 1999.

[43] S. Chulani, B. Boehm and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 573-583, 1999.

[44] K. Ganesan, T. Khoshgoftaar, and E. Allen, "Cased-based software quality prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol.10, no.2, pp. 139-152, 2000.

[45] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653-661, 2000.

[46] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, Wiley, New York, 2001.

[47] J.T. Connor, R.D. Martin, and L.E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, pp. 240-253, 1994.

[48] M. Han, J. Xi, S. Xu, and F.L. Yin, "Prediction of chaotic time series based on the recurrent predictor neural network," *IEEE Transactions on Signal Processing*, vol. 52, pp. 3409-3416, 2004.

[49] O. Renaud, J.L. Starck, and F. Murtagh, "Wavelet-based combined signal filtering and prediction," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 35, pp. 1241-1251, 2005.

[50] Y. Luo, T. Bergander, and A. Ben Hamza, "Anisotropic Laplace trend to enhance software reliability growth modelling," *Proc. International Conference on Modelling and Simulation*, Montréal, Canada, May 2006.

[51] Y. Luo, T. Bergander, and A. Ben Hamza, "Software reliability growth modelling using a weighted Laplace test statistic," *Proc. IEEE International Computer Software and Applications Conference*, Beijing, China, July 2007.

[52] T. Bergander, Y. Luo, and A. Ben Hamza, "Software defects prediction using operating characteristic curves," *Proc. IEEE International Workshop on Software Stability at Work*, Las Vegas, USA, August 2007.