Springer Series in Reliability Engineering

Shigeru Yamada
Yoshinobu Tamura

OSS Reliability Measurement and Assessment

Springer

Springer Series in Reliability Engineering

Series editor

Hoang Pham, Piscataway, USA

More information about this series at http://www.springer.com/series/6917

Shigeru Yamada · Yoshinobu Tamura

OSS Reliability Measurement and Assessment

Springer

Shigeru Yamada
Department of Social Management Engineering
Tottori University
Tottori
Japan

Yoshinobu Tamura
Yamaguchi University
Ube
Japan

ISSN 1614-7839 ISSN 2196-999X (electronic)
Springer Series in Reliability Engineering
ISBN 978-3-319-31817-2 ISBN 978-3-319-31818-9 (eBook)
DOI 10.1007/978-3-319-31818-9

Library of Congress Control Number: 2016934956

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Preface

We have been energetically proposed several reliability assessment methods for an open-source software in the last decade. Its measurement and management technologies for open-source software are essential to produce and maintain quality/reliable system by using open-source software. This book will serve as a textbook and reference book for graduate students and researchers in reliability for open-source software and modeling. Several methods of reliability assessment for open-source software are introduced. Our aim is to present state of the art of open-source software reliability measurement and assessment based on the stochastic modeling approach and recent research on this subject over the last 10 years. For example, the component-oriented reliability analysis based on AHP, ANP, and NHPP models, the stochastic differential equation models, and the hazard rate models are presented.

Chapter 1 introduces several aspects of software reliability, open-source software development paradigm, and its applications. Also, the basic concept of a mathematical model based on probability theory and statistics required to describe the software fault-detection or the software failure-occurrence phenomena and to estimate the software reliability quantitatively are introduced. Chapters 2 and 3 describe several methods of component-oriented reliability assessment based on AHP and ANP in order to consider the effect of each software component on the reliability of entire system under the distributed open-source development paradigm.

In case of the open-source software, it is necessary to grasp the situation of registration for bug tracking system, the degree of maturation of open-source software, and so on. Chapters 4–7 describe the methods of reliability assessment based on several stochastic models considering such characteristics from the standpoint of software reliability growth models. Chapters 8−10 provide several software tools for open-source software reliability assessment. These tools will be useful for software managers and engineers to assess the reliability of open-source software and embedded system software. Chapter 11 shows several numerical examples for the proposed methods of OSS reliability assessment based on several

actual data sets reported in the bug tracking systems for open-source projects. Also, Chap. 12 as the final part presents the performance illustrations for the developed tools.

We would like to express our sincere appreciation to Prof. Hoang Pham, Rutgers University, and editor Anthony Doyle, Springer-Verlag, London, for providing us with an opportunity to author this book.

Tottori, Japan Shigeru Yamada
February 2016

Contents

1 Software Reliability 1
1.1 Introduction 1
1.2 Definitions 3
1.3 Software Reliability Measurement and Assessment 5
1.4 Open Source Software Reliability 9
1.4.1 Brief Summary of Open Source Software 9
1.4.2 The Characteristics of OSS 10
1.4.3 Development Paradigm of OSS 11
References 13

2 NHPP Model and AHP for OSS Reliability Analysis 15
2.1 Component Reliability Analysis of OSS 15
2.1.1 Reliability Assessment Based on SRGM 15
2.1.2 Exponential SRGM 16
2.1.3 Inflection S-Shaped SRGM 16
2.1.4 Goodness-of-Fit Evaluation Criteria for Applied Model 16
2.1.5 Weight Parameter for Each Component Based on AHP 17
2.2 Reliability Analysis for Entire OSS System 19
2.2.1 Logarithmic Execution Time Model 19
2.2.2 Reliability Assessment Measures 19
References 20

3 NHPP Model and ANP for OSS Reliability Analysis 21
3.1 Reliability Assessment for Each Software Component 21
3.1.1 Reliability Assessment Based on SRGM 21
3.1.2 Exponential SRGM 22
3.1.3 Inflection S-Shaped SRGM 22

3.1.4 Goodness-of-Fit Evaluation Criteria for Applied Model . . . 22
3.1.5 Weight Parameter for Each Component Based on ANP . . . 23
3.2 Reliability Assessment for Entire System . . . 24
3.2.1 Inflection S-Shaped SRGM . . . 24
3.2.2 Software Reliability Assessment Measures . . . 25
References . . . 25

4 Stochastic Differential Equation Models for OSS Reliability Analysis . . . 27
4.1 Introduction . . . 27
4.2 Stochastic Differential Equation Modeling . . . 28
4.3 Method of Maximum-Likelihood . . . 29
4.4 Software Reliability Assessment Measures . . . 30
4.4.1 Expected Numbers of Detected Faults and Their Variances . . . 30
4.4.2 Mean Time Between Software Failures . . . 31
4.4.3 Coefficient of Variation . . . 32
References . . . 32

5 Hazard Rates for Embedded OSS Reliability Analysis . . . 33
5.1 Introduction . . . 33
5.2 Flexible Hazard Rate Model for Embedded OSS . . . 34
5.3 Reliability Assessment Measures . . . 35
References . . . 36

6 Reliability Analysis for Open Source Solution . . . 39
6.1 Introduction . . . 39
6.2 Stochastic Differential Equation Model . . . 40
6.3 Method of Maximum-Likelihood . . . 41
6.4 Software Reliability Assessment Measures . . . 42
6.4.1 Expected Numbers of Detected Faults . . . 42
6.4.2 Mean Time Between Software Failures . . . 43
6.4.3 Coefficient of Variation . . . 43
References . . . 43

7 Reliability Analysis for Mobile OSS . . . 45
7.1 Introduction . . . 45
7.2 Hazard Rate Model Depending on the Change of Network Traffic . . . 46
7.3 Reliability Assessment Measures . . . 47
7.4 Estimation of Network Traffic Density . . . 49
7.5 Parameter Estimation . . . 51
References . . . 52

8 Reliability Analysis Tool for Embedded OSS 53
8.1 Introduction 53
8.2 Hazard Rate Model for Embedded OSS Porting Phase 54
8.3 Reliability Assessment Measures 55
8.4 Optimal Release Problem for the Porting-Phase 56
8.5 Reliability Assessment Measures 57
8.6 Reliability/Portability Assessment Tool 59
8.6.1 Specification Requirement 59
8.6.2 Software Reliability Assessment Procedures 61
References 62

9 Reliability Analysis Tool for Open Source Solution 65
9.1 Introduction 65
9.2 Stochastic Differential Equation Modeling 66
9.3 Method of Maximum-Likelihood 68
9.4 Software Reliability Assessment Measures 69
9.5 Comparison of Goodness-of-Fit 69
9.6 Procedures of Reliability Analysis 70
9.7 Reliability Analysis Tool 71
9.7.1 Specification Requirement 71
9.7.2 Software Reliability Assessment Procedures 71
References 74

10 Reliability Analysis Tool for Mobile OSS 75
10.1 Introduction 75
10.2 Hazard Rate Model for Mobile OSS 77
10.3 Reliability Assessment Measures 78
10.4 Parameter Estimation 78
10.5 AIR Application for Reliability Analysis Considering the User Experience Design 79
10.5.1 Specification Requirement 79
10.5.2 User Experience Design 80
References 80

11 Actual Data and Numerical Examples of OSS Reliability Assessment 83
11.1 NHPP Model Based on AHP 83
11.1.1 Reliability Assessment for Each Component 83
11.1.2 Reliability Assessment for Entire System 108
11.1.3 Discussion for the Method of Reliability Assessment Based on AHP 111
11.2 NHPP Model Based on ANP 111
11.2.1 Reliability Assessment for Each Component 111
11.2.2 Reliability Assessment for Entire System 113
11.2.3 Discussion for the Method of Reliability Assessment Based on ANP 114

11.3 Stochastic Differential Equation Models 115
11.3.1 Data for Numerical Illustrations 115
11.3.2 Reliability Assessment 115
11.3.3 Sensitivity Analysis in Terms of Model Parameters . . . 115
11.3.4 Results of Goodness-of-Fit Comparison. 117
11.3.5 Discussion for the Method of Reliability Assessment Based on Stochastic Differential Equation Models 125
11.4 Hazard Rates for Embedded OSS. 126
11.4.1 Embedded OSS 126
11.4.2 Reliability Assessment 126
11.4.3 Comparison of Goodness-of-Fit 131
11.4.4 Prediction Accuracy After the End of Fault-Report 132
11.4.5 Optimal Software Release Problem for the Porting-Phase of Embedded OSS 133
11.4.6 Discussion for the Method of Reliability Assessment Based on Hazard Rates for Embedded OSS. 136
11.5 Applied Examples for Open Source Solution 137
11.5.1 Data for Numerical Illustrations 137
11.5.2 Reliability Assessment 138
11.5.3 Discussion for the Method of Reliability Assessment on the Open Source Solution 144
References 144

12 Performance Illustrations of Software Tool 145
12.1 Embedded OSS 145
12.1.1 Applied Data 145
12.1.2 Optimal Release Problem with Reliability of Embedded OSS 152
12.1.3 Discussion for the Software Tool 154
12.2 Open Source Solution 154
12.2.1 Applied Data 154
12.2.2 Discussion for the Software Tool 155
12.3 Mobile OSS 162
12.3.1 Applied Data 162
12.3.2 Discussion for the Software Tool 176
References 181

13 Exercises 183
13.1 Exercise 1 183
13.2 Exercise 2 183
13.3 Exercise 3 183
13.4 Exercise 4 184

Chapter 1
Software Reliability

1.1 Introduction

In recent years, many computer system failures have been caused by software faults which were introduced during the software development process. This is an inevitable problem since a software system installed in the computer system is an intellectual product consisting of documents and source programs developed by human activities. Then, total quality Management (TQM) is considered to be one of the key technologies to produce more highly reliable software products. In case of TQM for software management, all phase of the development process, i.e. specification, design, coding, and testing, have to be controlled systematically to prevent software fault-introduction as much as possible and to detect the introduced faults in the software system as early as possible. Basically, the concept of TQM means to assure the quality of the intermediate products in each phase to the next phase. Particularly, quality control executed at the testing phase which is the last stage of the software development process is very important. During the testing phase, the product quality and the software performance during the operation phase are evaluated and assured. Concretely, a lot of software faults introduced in the software system through the first three phases of the development process by human activities are detected, corrected, and removed. Figure 1.1 shows a general software development process so-called a water-fall diagram.

Under a software development process based on TQM, it is necessary to define and measure the software quality. Generally, software quality realized through the development process can be distinguished between product quality and process quality. The former is the attributes of the products resulting from the software development. This includes the clarity of the specification documents, the design documents, and the source code, the traceability, the reliability, and the test coverage. The latter is the attributes contributed by the software development environment. This includes the rigor of the production technology, the productivity of the development tools, the ability or skill of the development persons, the communicativeness among the development team, and the availability of the development facilities. From the

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_1

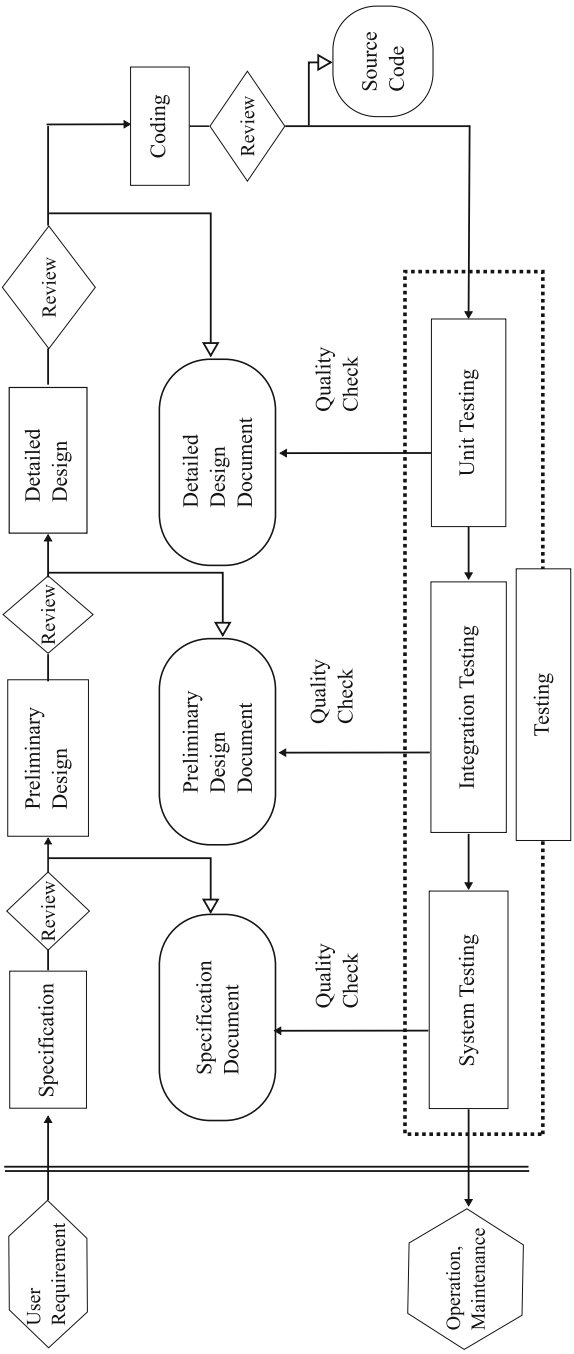

Fig. 1.1 A general software development process (water-fall paradigm)

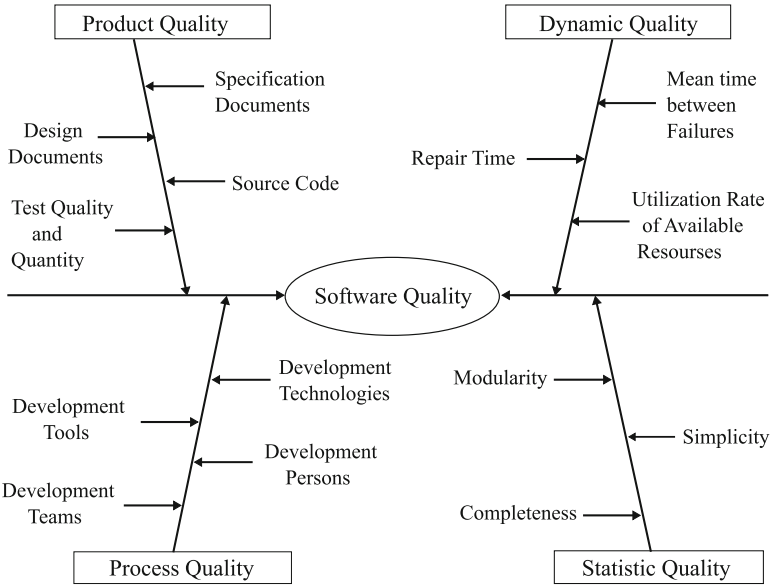

Fig. 1.2 The elements of software quality based on a cause-and-effect diagram

measurement point of view, software quality can be distinguished between dynamic quality and static quality. The former is the attributes confirmed by executing the program on the computers and examining the dynamic behavior. The latter is the attributes validated by reviews and inspections at each phase without executing the program. These elements of software quality are summarized by a cause-and-effect diagram in Fig. 1.2.

Recently, recognizing the importance of standardizing quality characteristics, ISO (International Standard Organization) has studied and summarized six quality characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. Particularly, the characteristic of reliability is important as "must-be quality." From the view points of product quality and dynamic quality above, *software reliability* is defined as the probability that a software will not cause the failure of a system for a specified time under specified conditions.

1.2 Definitions

We compare the characteristics of software reliability with those of hardware reliability as follows:

Software Reliability

1. Software failures can be due to no wear-out phenomenon.
2. Software reliability is, inherently, determined during the earlier phase of the development process, i.e., specification and design phases.
3. Software reliability cannot be improved by redundancy with identical versions.
4. A verification method of software reliability has not been established.
5. A maintenance technology is not established since the market of software products is rather recent.

Hardware Reliability

1. Hardware failures can be due to wear.
2. Hardware reliability is affected by deficiencies injected during all phases of the development, operation, and maintenance.
3. Hardware reliability can be improved by redundancy with identical units.
4. A testing method of hardware reliability is established and standardized.
5. A maintenance technology is advanced since the market of hardware products is established and the user environment is seized.

Generally, a software failure caused by software faults latent in the software system cannot occur expected for a special occasion when a set of special data is put into the system under a special condition, i.e. the program path including software faults is executed. Therefore, the software reliability is determined by the input data and the internal condition of the program. We summarize the definitions of the terms related to the software reliability in the following.

A software system is a product which consists of the programs and documents as a result of the software development process discussed in previous section. Specification derived by analyzing user requirements for the software system is a document which describes the expected performance of the system. When the software performance deviates from the specification and the output variable has an improper value or the normal processing is interrupted, it is said that a software failure occurs. That is, *software failure* is defied as a departure of program operation from the program requirements. The cause of software failure is called a software fault. Then, *software fault* is defined as a defect in the program which causes a software failure. The software fault is usually called software bug. *Software error* is defined as human action that results in the software system containing a software fault. Thus, the software fault is considered to be a manifestation of software errors.

Based on the basic definitions above, we can describe the software behavior as Input (I)-Program (P)-Output (O) model as shown in Fig. 1.3. In this model, the program is considered as a mapping from the input space constituting input data available on use to the output space constituting output data or interruptions of normal processing. Testing space T is an input subspace of I, of which the performance can be verified and validated by software testing. Software faults detected and removed during the testing phase map the elements of input subspace E into an output subspace O' constituting the events of a software failure. That is, the faults detected during

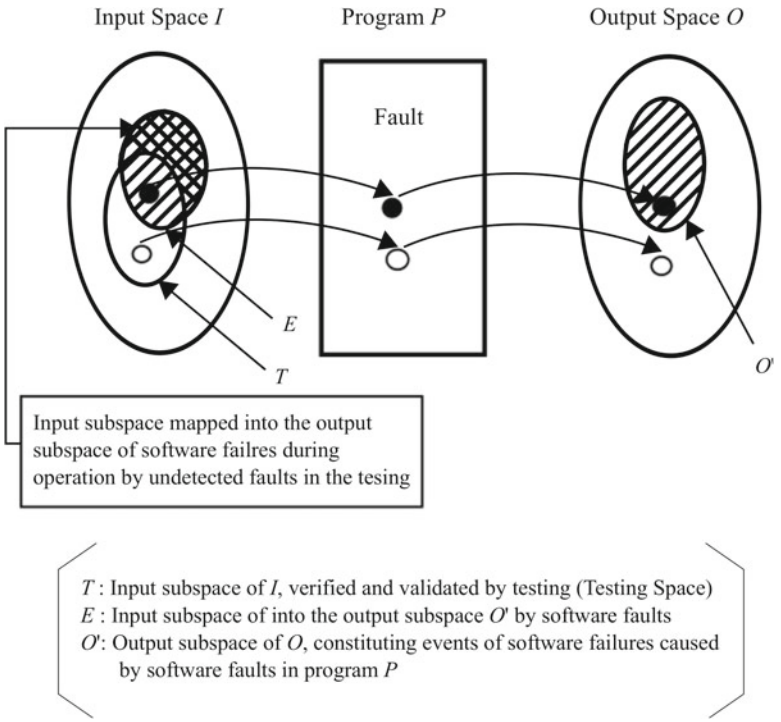

Fig. 1.3 An Input-Program-Output model for software behavior

the testing phase belong to the intersection of subspaces E and T. Software faults remaining in the operation phase belong to the subspace E but not to the testing space T.

In this chapter, software error may be used to mean a defect detected and corrected during the testing or operation phase without distinction from software fault.

1.3 Software Reliability Measurement and Assessment

Generally, a mathematical model based on probability theory and statistics is useful to describe the software fault-detection phenomena or the software failure-occurrence phenomena and estimate the software reliability quantitatively. During testing phase in the software development process, software faults are detected and removed with a lot of test-effort expenditures. Then,the number of remaining faults in the software system is decreasing as the testing goes on. This means that the probability of software failure-occurrence is decreasing so the software reliability is increasing and the time-interval between software failures becomes longer with the testing

time. A mathematical tool which treats such software reliability aspect is a *software reliability growth model* [1].

Based on the definitions discussed previous section, we can make a software reliability growth model based on the assumptions for actual environments during the testing phase or the operation phase. Then, we can define the following random variables on the number of detected faults and the software failure-occurrence time.

$N(t)$ the cumulative number of detected software faults (or the cumulative number of observed software failures) up to testing time t,
S_i the ith software failure occurrence time ($i = 1, 2, \ldots;\ S_0 = 0$),
X_i the time-interval between $(i-1)$st and ith software failures ($i = 1, 2, \ldots;\ X_0 = 0$).

Figure 1.4 shows occurrence of event $\{N(t) = i\}$ since i faults have been detected up to time t. From these definitions, we have

$$S_i = \sum_{k=1}^{i} X_k, \quad X_i = S_i - S_{i-1}. \tag{1.1}$$

Assuming that the hazard rate, i.e. the *software failure rate*, for X_i ($i = 1,\ 2,\ \ldots$), $z_i(x)$, is proportional to the current number of residual faults remaining in the system, we have

$$z_i(x) = (N - i + 1)\lambda(x), \quad i = 1,\ 2,\ \ldots,\ N;\ x \geq 0,\ \lambda(x) > 0, \tag{1.2}$$

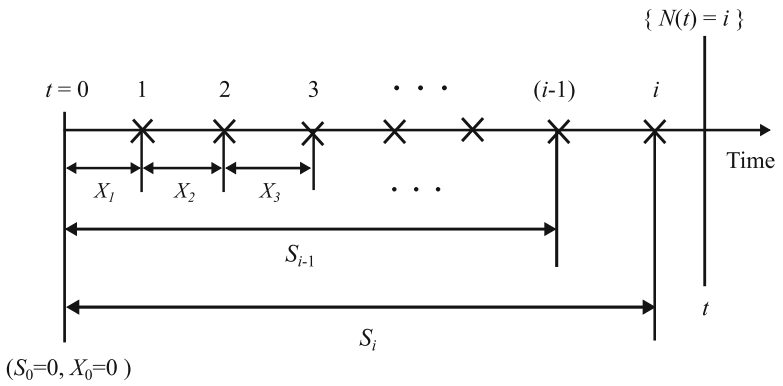

Fig. 1.4 The stochastic quantities related to a software fault detection phenomenon or a software failure occurrence phenomenon.

where N is the initial fault content and $\lambda(x)$ the software failure rate per fault remaining in the system at time x. If we consider two special cases in Eq. (1.2) as

$$\lambda(x) = \phi, \quad \phi > 0, \tag{1.3}$$

$$\lambda(x) = \phi x^{m-1}, \quad \phi > 0, \quad m > 0, \tag{1.4}$$

then two typical software hazard rate models, respectively called the Jelinski–Moranda model and the Wagoner model can be derived, where ϕ and m are constant parameters. Usually, it is completely fault free or failure free. Then, we have a software hazard rate model called the Moranda model for the case of the infinite number of software failure occurrences as

$$z_i(x) = Dk^{i-1}, \quad i = 1, 2, \ldots; \; D > 0, \quad 0 < k < 1, \tag{1.5}$$

where D is the initial software hazard rate and k the decreasing ratio, Eq. (1.5) describes a software failure-occurrence phenomenon where a software system has high frequency of software failure occurrence during the early stage of the testing or the operation phase and it gradually decreases thereafter, Based on the software hazard rate models above,we can derive several software reliability assessment measures. For example,the software reliability function for X_i ($i = 1, 2, \ldots$) is given as

$$R_i(x) = \exp\left[-\int_0^x z_i(x)dx\right], \quad i = 1\,, 2, \ldots. \tag{1.6}$$

Further, we also discuss NHPP models, which are modeled for random variable $N(t)$ as typical software reliability growth models, In the NHPP models,a nonhomogeneous Poisson process (NHPP) is assumed for the random variable $N(t)$,the distribution function of which is given by

$$\begin{aligned} \Pr\{N(t) = n\} &= \frac{H(t)^n}{n!}\exp[-H(t)], \quad n = 1, 2, \ldots, \\ H(t) \equiv \mathrm{E}[N(t)] &= \int_0^t h(x)dx, \end{aligned} \tag{1.7}$$

where $\Pr[\cdot]$ and $\mathrm{E}[\cdot]$ mean the probability and expectation,respectively. $H(t)$ in Eq. (1.7) is called a mean value function which indicates the expectation of $N(t)$, i.e., the expected cumulative number of faults detected (or the expected cumulative number of software failures occurred) in the time interval $(0, t]$, and $h(t)$ in Eq. (1.7) called an intensity function which indicated the instantaneous fault-detection rate at time t. From Eq. (1.7), various software reliability assessment measures can be derived. For examples, the expected number of faults remaining in the system at time t is given by

$$n(t) = a - H(t), \tag{1.8}$$

where $a \equiv H(\infty)$, i.e., parameter a denotes the expected initial fault content in the software system. Given that the testing or the operation has been going on up to time t, the probability that a software failure does not occur in the time-interval $(t, t+x](x >= 0)$ is given by conditional probability $\Pr\{X_i > x | S_{i-1} = t\}$ as

$$R(x|t) = \exp[H(t) - H(x+t)], \quad t \geq 0,\ x \geq 0. \tag{1.9}$$

$R(x|t)$ in Eq. (1.9) is a so-called software reliability. Measures of MTBF (mean time between software failures or fault detections) can be obtained follows:

$$MTBF_I(t) = \frac{1}{h(t)}, \tag{1.10}$$

$$MTBF_C(t) = \frac{t}{H(t)}. \tag{1.11}$$

MTBFs in Eqs. (1.10) and (1.11) are called instantaneous MTBF and cumulative MTBF, respectively. It is obvious that the lower the value of $n(t)$ in Eq. (1.8), the higher the value $R(x|t)$ for specified x in Eq. (1.9), or the longer the value of MTBFs in Eqs. (1.10) and (1.11), the higher the achieved software reliability is. Then, analyzing actual test data with accepted NHPP models, there measures can be utilized to assess software reliability during the testing or operation phase, where statistical inferences, i.e. parameter estimation and goodness-of-fit test, are usually performed by a method of maximum-likelihood.

To assess the software reliability actually, it is necessary to specify the mean value function $H(t)$ in Eq. (1.7). Many NHPP models considering the various testing or operation environments for software reliability assessment have been proposed in the last decade. As discussed above, a software reliability growth is described as the relationship between the elapsed testing or operation time and the cumulative number of detected faults and can be shown as the reliability growth curve mathematically.

Among the NHPP models, exponential and modified exponential software reliability growth models are appropriate when the observed reliability growth curve shows an exponential curve. Similarly, delayed S-shaped and inflection S-shaped software reliability growth models are appropriate when the reliability growth curve is S-shaped.

In addition, as for computer makers or software houses in Japan, logistic curve and Gompertz curve models have often been used as software quality assessment models, on the assumption that software fault-detection phenomena can be shown by S-shaped reliability growth curves. In these deterministic models, the cumulative number of faults detected up to testing t is formulated by the following growth equations:

$$L(t) = \frac{k}{1 + me^{-\alpha t}}, \quad m > 0,\ \alpha > 0,\ k > 0, \tag{1.12}$$

$$G(t) = ka^{(b^t)}, \quad 0 < a < 1,\ 0 < b < 1,\ k > 0. \tag{1.13}$$

In (1.12) and (1.13), assuming that a convergence value of each curve ($L(\infty)$ or $G(\infty)$), i.e., parameter k, represents the initial fault content in the software system, it can be estimated by a regression analysis.

1.4 Open Source Software Reliability

1.4.1 Brief Summary of Open Source Software

All over the world people can obtain the information at the same time by growing rate of Internet access around the world in recent years. In accordance with such a penetration of the Internet, it is increasing public awareness of the importance of online real-time and interactive functions. Therefore, the distributed software development paradigm based on the information technologies are expanding at an explosive pace. Especially, new development paradigm such as client/server system, web programming, object-oriented development, and also on distributed development paradigm by using network technologies have been in heavy usage by the software developers [2–5].

At present, there are a number of development paradigms within the same company, the development environment joined hands with several software company, the collaborative development based on consortia formation in terms of the distributed development environment. Furthermore, a software development paradigm based on an open source project is rapidly spreading [6, 7].

The open source project contains special features so-called software composition that the geographically-dispersed several components are developed in all parts of the world. The successful experience of adopting the distributed development model in open source projects includes GNU/Linux operating system,[1] Apache Web server, and so on.[2] Especially, OSS (open source software) is frequently applied as server use, instead of client use. Then, such an open source system development is still ever-expanding now.

A computer-software is developed by human work, therefore many software faults must be introduced into the software product during the development process. Thus, these software faults often cause complicated break-downs of computer systems. Recently, it becomes more difficult for the developers to produce highly-reliable software systems efficiently because of the diversified and complicated software requirements. Therefore, it is necessary to control the software development process in terms of quality and reliability. Note that a *software failure* is defined as an unacceptable departure of program operation caused by a *software fault* remaining in the software system as mentioned in Sect. 1.2. Basically, software reliability can be evaluated by the number of detected faults or the software failure-occurrence time. Especially,

[1] Linux is a Registered Trademark of Linus Torvalds.

[2] Other company, product, or service names may be trademarks or service marks of others.

software reliability models which can describe software fault-detection or failure-occurrence phenomena are called *software reliability growth models* (SRGM's) [1]. The SRGM's are useful to assess the reliability for quality control and testing-process control of software development.

On the other hand, the effective testing management method for distributed development environment as typified by open source project has only a few presented [8–11]. In case of considering the effect of a debugging process on entire system in the development of a method of reliability assessment for distributed development environment, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of component, skill of fault reporter, and so on.

1.4.2 The Characteristics of OSS

At present, OSS (Open Source Software) systems serve as key components of critical infrastructures in the society. OSS projects possess a unique feature known as software composition by which components are developed by teams that are geographically-dispersed throughout the word. Successful OSS projects includes Apache HTTP server, MySQL database server, OpenStack cloud software, Firefox Web browser, and GNU/Linux operating system. However, poor handling of quality issues and customer support has limited the progress of OSS, because the development cycle of OSS has no testing phase. For example, mobile OSS has been gaining a lot of attention in the embedded system area, i.e., Android, BusyBox, Firefox OS, etc. However, the installer software developed under the third-party developers indirectly effects the reliability of a mobile device. Therefore, it is difficult for many companies to assess the reliability of a mobile OSS, because a mobile OSS includes several software versions. Another closely related issues is that of software vulnerability posed by the open source nature of the code, raising the possibility of security loopholes. For the above mentioned reasons, it is difficult for software managers to assess the reliability of OSS.

We compare the characteristics of software development under the OSS and the proprietary software paradigms as follows:

OSS

1. The specification continuously changes with each version upgrade.
2. OSS has no specific testing phase.
3. Several versions of OSS are uploaded to the website of the OSS project.
4. It is difficult to clearly distinguish between developers and users.
5. Many OSS possess different licenses.

Proprietary Software

1. The specification is fixed in the initiation phase.
2. The proprietary software has a specific testing phase.
3. The delivery of software is specified as is the software release time.

4. The user cannot access the source code. The maintenance is performed by the software engineers.
5. The user of proprietary software must contract with the software development company.

In particular, OSS have several licenses as follows:

1. GPL (GNU General Public License).
2. LGPL (GNU Lesser General Public License).
3. BSD License.
4. X11 License.
5. Apache Software License.

There are many software licenses in a variety of OSS project areas. Therefore, it is known that it is difficult for software managers to use the OSS for the commercial software.

1.4.3 Development Paradigm of OSS

At present, many OSS are developed under several open source projects. For example, the Firefox Web browser, Firefox OS, and Thunderbird mailer are developed and managed under the Mozilla.org project. Also, Apache HTTP server, Tomcat, and Flex are developed and managed under the Apache software foundation. These open source projects control the development phase of many open source projects. The typical development cycle of open source projects is shown in Fig. 1.5. On the other hand, Fig. 1.6 shows the water fall model of typical software development. Figures 1.5 and 1.6 show that the open source project has no the specific testing phase. Also, the specification of OSS continuously changes with the version upgrade, the because the software of several versions are uploaded to the website of OSS project.

It is difficult for software managers to assess OSS reliability because of the differences among the development style of OSS and traditional software. Therefore, it

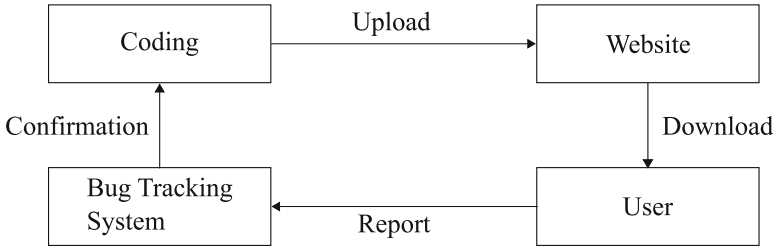

Fig. 1.5 The development cycle of open source project

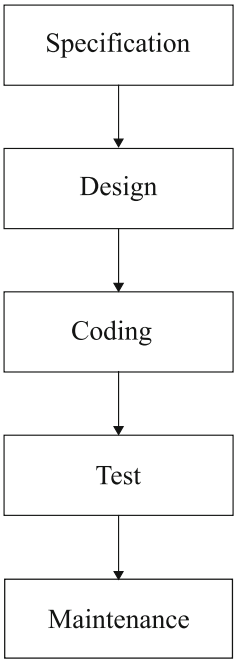

Fig. 1.6 The water fall model of typical software development

is important to assess and manage considering the characteristics of the OSS development paradigm.

In particular, OSS have several versions of the development process as follows:

1. Bug fix version (most urgent issue such as patch).
2. Minor version (minor revision by the addition of a component and module).
3. Major version (significant revision for specification).

Also, the version number of OSS is generally described as the "(Major version number. Minor version number. Revision number. Build number)", e.g., (2.1.2103.1104). There are several versions for each OSS. Therefore, it is known that it is difficult for software managers to select the appropriate OSS, because several OSS are uploaded to the website of open source project.

This book focus on the reliability of OSS with the above mentioned characteristics. Several methods of reliability assessment for OSS are presented in this book. Also, several numerical examples for each method of reliability assessment are given by using actual fault data of OSS. These method introduced in this book may be useful for software managers to assess the reliability of a software system developed using the OSS paradigm.

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 4th edn. (McGraw-Hill, New York, 1982)
3. A. Umar, *Distributed Computing and Client-Server Systems* (Prentice Hall, New Jersey, 1993)
4. L.T. Vaughn, *Client/Server System Design and Implementation* (McGraw-Hill, New York, 1994)
5. M. Matsumoto, M. Koyamada, T. Matsuodani, *Technique of Software Development Verification* (in Japanese) (IEICE, Tokyo, 1997)
6. M. Takahashi, *The Method of Effort Estimation under Client/Server System Development: Models and Applications* (in Japanese) (Soft Research Center, Tokyo, 1998)
7. T. Akahane, *Testing Method of Client/Server System* (in Japanese) (Soft Research Center, Tokyo, 1998)
8. A. MacCormack, J. Rusnak, C.Y. Baldwin, Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Inf. J. Manag. Sci. **52**(7), 1015–1030 (2006)
9. G. Kuk, Strategic interaction and knowledge sharing in the KDE developer mailing list. Inf. J. Manag. Sci. **52**(7), 1031–1042 (2006)
10. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. Proc. Workshop Open Source Softw. Eng. (WOSSE) **30**(4), 67–72 (2005)
11. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering* (*FSE-12*) (2004), pp. 263–272

Chapter 2
NHPP Model and AHP for OSS Reliability Analysis

In this chapter, we focus on the X desktop environment of which the product is the software system developed under the open source project. There are many X desktop environments operating on UNIX-like operating systems which are known as free softwares. Such an X desktop environment is known as the software which offer the interface, design, operationality to users. The purpose of X desktop environment is to improve operationality by providing the graphical user interfaces on UNIX-like operating systems. At present, the representative examples include GNOME and KDE. Especially, GNOME and KDE have the characteristics of flexible and easy-to-use GUI and a long-use history. On the other hand, these are pointing out as the hefty operationality, a lot of library, complexities of dependence, and so on.

In order to consider the effect of each software component on the reliability of a system developed in a distributed environment, we apply the AHP which well established decision making methods. Moreover, we propose the method of reliability assessment based on an SRGM incorporating the interaction among each software component. We also show several numerical examples of software reliability assessment by using the actual data.

2.1 Component Reliability Analysis of OSS

2.1.1 Reliability Assessment Based on SRGM

Many SRGM's have been used as a conventional methods to assess the reliability, quality control, and testing-process control of software development. Among others, nonhomogeneous Poisson process (NHPP) models have been discussed by many researchers, since these NHPP models can be easily applied during software development. In this section, we discuss NHPP models to analyze software fault-detection count data.

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_2

According to the growth curve of the cumulative number of detected faults, we assume that the software reliability in each software component is assessed by applying the following SRGM's based on the NHPP [1]:

- Exponential SRGM.
- Inflection S-shaped SRGM.

The NHPP models have been discussed by many researchers, since the models can be easily applied in actual software projects. Moreover, we apply the method of maximum-likelihood to estimate the model parameters. Below are the expressions for various software reliability assessment measures from the NHPP models.

2.1.2 Exponential SRGM

The mean value function of the exponential SRGM is given as follows:

$$E_i(t) = a_i(1 - e^{-b_i t}) \quad (a_i > 0,\ b_i > 0), \tag{2.1}$$

where $E_i(t)$ represents the expected cumulative number of faults detected up to the module testing time t ($t \geq 0$) is mean value functions for the ith software component. In Eq. (2.1), $a_i (i = 1, 2, \ldots, n)$ is the expected number of initial inherent faults for the ith software component, and $b_i (i = 1, 2, \ldots, n)$ the software failure rate per inherent fault for the ith software component.

2.1.3 Inflection S-Shaped SRGM

The mean value function of the inflection S-shaped SRGM is given as follows:

$$D_i(t) = \frac{a_i(1 - e^{-b_i t})}{(1 + c_i \cdot e^{-b_i t})} \quad (a_i > 0,\ b_i > 0,\ c_i > 0), \tag{2.2}$$

where $a_i (i = 1, 2, \ldots, n)$ is the expected number of initial inherent faults for the ith software component, and $b_i (i = 1, 2, \ldots, n)$ the software failure rate per inherent fault for the ith software component. Moreover, $c_i (i = 1, 2, \ldots, n)$ represents the inflection rate for the ith software component.

2.1.4 Goodness-of-Fit Evaluation Criteria for Applied Model

In this chapter, we compare the model goodness-of-fit of two conventional SRGM's for the observed data set. We use the following goodness-of-fit evaluation criteria,

i.e., the Akaike's information criterion (AIC) and the mean square error (MSE). Suppose that K data pairs $(t_k, y_k)(k = 1, 2, \ldots, K)$ are observed during the system testing-phase, where y_k is the cumulative number of software failures observed in the time interval $(0, t_k]$.

(A) AIC
AIC helps us to select the optimal model among ones estimated by the method of maximum-likelihood. It is given by

$$\begin{aligned} \text{AIC} = \; & -2 \cdot (\text{the logarithmic maximum-} \\ & \text{likelihood}) + 2 \cdot (\text{the number of} \\ & \text{free model parameters}). \end{aligned} \tag{2.3}$$

Differences among AIC values are significant, not the value themselves. A model possessing the smallest AIC best a data set when the difference between two models is greater than or equal to 1. However, there are no significant difference among two models in the case where the differences of AIC's are less than 1.

(B) MSE
The mean square error can be obtained by dividing the sum of square errors between the observed value, y_k, and its estimate, $\hat{y}_k$, by the number of pairs of data, n. That is,

$$\text{MSE} = \frac{1}{n} \sum_{k=1}^{n} (y_k - \hat{y}_k)^2. \tag{2.4}$$

$\hat{y}_k$ in Eq. (2.4) is obtained from $\hat{y}_k = \hat{H}(t_k)(k = 1, 2, \ldots, n)$. A small mean square error indicates that the selected model fits the observed well.

We compare the two conventional SRGM's by using the above-described goodness-of-fit evaluation criteria. Concretely speaking, AIC is the first goodness-of-fit evaluation criterion, and MSE is the secondary goodness-of-fit measure, i.e., we select the appropriate model when the difference of value in AIC are greater than or equal to 1, otherwise we select the appropriate model based on the value of MSE.

2.1.5 Weight Parameter for Each Component Based on AHP

The AHP developed in the 1970s is utilized widely in Europe and the United States for management issues, energy problems, decision-making, urban planning. The AHP is considered to be one of the most effective methods for decision-making support [2, 3].

When considering the effect of debugging process on an entire system in the development of a software reliability assessment method for distributed development environment, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of a component, skill of fault reporter, and so on.

Also, it is rare that collected data sets entirely contain all the information needed to assess software reliability, although these data sets for deeply-intertwined factors completely collected from the bug tracking system. Therefore, it is difficult to estimate the effect of each component on the entire system by using the collected data sets only.

In this chapter, we propose a reliability assessment method based on the AHP to estimate the effect of each component on the entire system in a complicated environment. Specifically, we can assess the importance level of faults detected for each component, the size of component, the skill of fault reporter and so on, as evaluation criteria for the AHP.

Let $w_i (i = 1, 2, \ldots, n)$ be the weight parameters for evaluation criteria of the AHP. Then, the pair comparison matrix is given as follows:

$$A = \begin{bmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \cdots & \frac{w_1}{w_n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \cdots & \frac{w_2}{w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \cdots & \frac{w_n}{w_n} \end{bmatrix}. \tag{2.5}$$

We can obtain the weight parameter α_i for each evaluation criterion from the above pair comparison matrix by using the following geometric average:

$$\begin{aligned} \alpha_i &= \sqrt[n]{\prod_{j=1}^{n} x_{ij}}, \\ x_{ij} &= \frac{w_i}{w_j}. \end{aligned} \tag{2.6}$$

Therefore, the total weight parameter for each evaluation criterion is given by the following equation:

$$\beta_i = \frac{\alpha_i}{\sum_{i=1}^{n} \alpha_i}. \tag{2.7}$$

By using the weight parameter β_i in Eq. (2.7), we can obtain the total weight parameter p_i which represents the level of importance for each component.

2.2 Reliability Analysis for Entire OSS System

2.2.1 Logarithmic Execution Time Model

The operating environment of OSS possesses characteristics of the susceptible to various application softwares. Therefore, it is different from a conventional software system developed by an identical organization. Then, the expected number of detected faults continues to increase from the effect of the interaction among application softwares, i.e., the number of detected faults can not converge to a fixed value.

As mentioned above, we apply the logarithmic Poisson execution time model based on the assumption that the number of detected faults is infinity. Thus, we consider the following structure of the mean value function $\mu(t)$ for the entire system because an NHPP model is characterized by its mean value function:

$$\mu(t) = \frac{1}{\theta - P} \ln[\lambda_0(\theta - P)t + 1] \quad (0 < \theta,\ 0 < \lambda_0,\ 0 < P < 1), \tag{2.8}$$

where λ_0 is the intensity of initial inherent failure, θ the reduction rate of the failure intensity rate per inherent fault. Moreover, we assume that the parameter P in Eq. (2.8) represents the following weighted average in terms of the weight parameter p_i estimated by the AHP and the software failure rate per inherent fault b_i for the ith software component in Eqs. (2.1) or (2.2):

$$P = \sum_{i=1}^{n} p_i \cdot b_i, \tag{2.9}$$

where n represents the number of software components and $p_i (i = 1, 2, \ldots, n)$ the weight parameter for each component.

2.2.2 Reliability Assessment Measures

We can give the following expressions as software reliability assessment measures derived from the NHPP model given by Eq. (2.8):

- *Software reliability*
  The software reliability can be defined as the probability that a software failure does not occur during the time-interval $(t, t+x](t \geq 0, x \geq 0)$ after the testing-time t. The software reliability is given by

$$R(x|t) = \exp[\mu(t) - \mu(t + x)], \quad (t \geq 0, x \geq 0). \tag{2.10}$$

- *Instantaneous mean time between software failures*
  The instantaneous mean time between software failures ($MTBF_I$) measures the frequency of software failure-occurrence, and is given by

$$MTBF_I(t) = \frac{1}{\frac{d\mu(t)}{dt}}. \tag{2.11}$$

- *Cumulative mean time between software failures*
  The cumulative mean time between software failures ($MTBF_C$) is given as follows:

$$MTBF_C(t) = \frac{t}{\mu(t)}. \tag{2.12}$$

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. T. Satty, *The Analytic Hierarchy Process* (McGraw-Hill, New York, 1980)
3. E. Kinoshita, *Introductory AHP (in Japanese)* (JUSE Press, Tokyo, 2000)

Chapter 3
NHPP Model and ANP for OSS Reliability Analysis

In this chapter, in order to consider the effect of each software component on the reliability of an entire system in a distributed development environment, we apply the ANP (Analytic Network Process) which is a popular decision-making method. Moreover, we propose a method of reliability assessment based on the SRGM incorporating the interaction among software components. Also, we discuss a method of reliability assessment for OSS projects as a typical case of a distributed development environment.

3.1 Reliability Assessment for Each Software Component

3.1.1 Reliability Assessment Based on SRGM

Many SRGM's have been used as the conventional methods to assess the reliability for quality control and testing-process control during software development. Among others, nonhomogeneous Poisson process (NHPP) models have been discussed by many researchers since the NHPP models can be easily applied during software development. In this chapter, we analyze software fault-detection count data based on an NHPP model.

According to the growth curve of the cumulative number of detected faults, we assume that the software reliability for each software component can be assessed by applying the following SRGM's based on NHPP [1]:

- Exponential SRGM
- Inflection S-shaped SRGM

The NHPP models have been discussed by many researchers since the models can be easily applied in the software development. Moreover, we apply the method of maximum-likelihood to estimate the unknown model parameters. We can give the expressions for various software reliability assessment measures from the NHPP models with specified mean value functions.

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_3

3.1.2 Exponential SRGM

The mean value function of the exponential SRGM is given as follows:

$$E_i(t) = a_i(1 - e^{-b_i t}) \quad (a_i > 0, b_i > 0), \tag{3.1}$$

where $E_i(t)$ is the mean value function for the ith software component, which is the expected cumulative number of faults detected up to the module testing-time $t\,(t \geq 0)$. In Eq. (3.1), $a_i\,(i = 1, 2, \ldots, n)$ is the expected number of initial inherent faults for the ith software component, $b_i\,(i = 1, 2, \ldots, n)$ the software failure rate per inherent fault for the ith software component.

3.1.3 Inflection S-Shaped SRGM

The mean value function of the inflection S-shaped SRGM is given as follows:

$$D_i(t) = \frac{a_i(1 - e^{-b_i t})}{(1 + c_i \cdot e^{-b_i t})} \quad (a_i > 0, b_i > 0, c_i > 0), \tag{3.2}$$

where $D_i(t)$ is the mean value function for the ith software component, which is the expected cumulative number of faults detected up to the module testing-time $t\,(t \geq 0)$. In Eq. (3.2), $a_i\,(i = 1, 2, \ldots, n)$ is the expected number of initial inherent faults for the ith software component, $b_i\,(i = 1, 2, \ldots, n)$ the software failure rate per inherent fault for the ith software component. Moreover, $c_i\,(i = 1, 2, \ldots, n)$ represents the inflection rate for the ith software component.

3.1.4 Goodness-of-Fit Evaluation Criteria for Applied Model

In this chapter, we compare the goodness-of-fit of models applied in Sects. 2.1.2 and 2.1.3 with existing two conventional SRGM's for the observed data set. We also use the following goodness-of-fit evaluation criteria, i.e., the Akaike's information criterion (AIC) and the mean square error (MSE).

- AIC
  AIC helps us to select the optimum model among models estimated by the method of maximum-likelihood.
- MSE
  MSE can be obtained by dividing the sum of square errors between the observed value, y_k, and its estimated one, $\hat{y}_k$, by the number of data pairs, n.

3.1.5 Weight Parameter for Each Component Based on ANP

To consider the effect of debugging process on entire system in the development of a software reliability assessment method for distributed development environment, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of a component, skill of fault reporter, and so on.

Also, it is difficult to consider that collected data sets entirely contain the information in terms of software reliability, although these data sets for deeply-intertwined factors completely collected from bug tracking system. Therefore, it is difficult to estimate the effect of each component on the entire system by using the collected data sets only.

In this chapter, we propose the reliability assessment method based on the ANP [2, 3] to estimate the effect of each component on the entire system in a complicated situation. Specifically, we apply the importance level of faults detected for each component, the skill of the fault reporter, and the skill of the fault repairer as evaluation criteria for the ANP.

Figure 3.1 shows the network structure in this chapter. "Severity", "Assigned to" and "Reporter" are evaluation criteria, whereas "general", "other", "xfce4", "xfdesktop", "xffm", and "xfwm" are components. The super-matrix is given as follows:

$$S = \begin{bmatrix} A_1 & 0 \\ B_{21} & A_2 \end{bmatrix} \tag{3.3}$$

$$\left(A_1 = 0, \quad B_{21} = \begin{bmatrix} v \\ 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & W \\ U & 0 \end{bmatrix} \right).$$

Then, A_i is the evaluation matrix in cluster i, and B_{21} the evaluation matrix from cluster 1 to cluster 2. Moreover, v is the level of importance of each component, U the

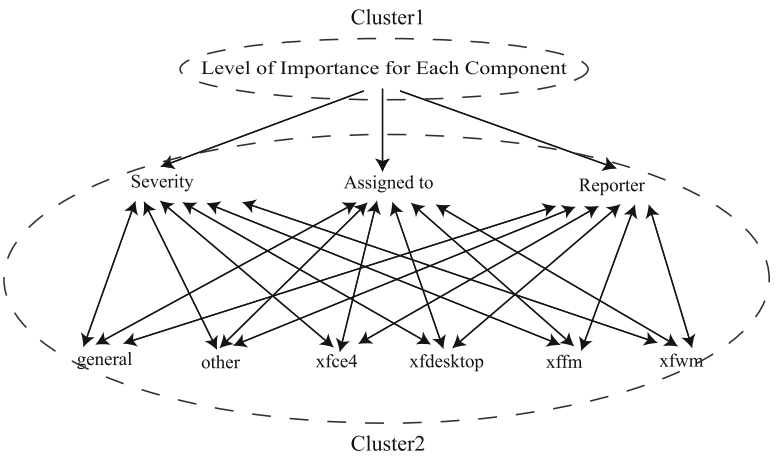

Fig. 3.1 Network structure of ANP

evaluation matrix which influences from each component to evaluation criteria, and W the evaluation matrix which influences from evaluation criteria to each component.

First, in order to standardize a matrix, the maximum eigenvalue λ_1 and λ_2 of the partial matrix in a diagonal block A_1 and A_2 is calculated as

$$\overline{A_i} = \frac{1}{\lambda_i} A_i \quad (i = 1, 2), \tag{3.4}$$

$$\overline{B_{21}} = \frac{1}{\lambda_1} B_{21}. \tag{3.5}$$

Then, let λ_i be 1 if $A_i = 0$. Then, the super-matrix is given as follows:

$$S = \left[\begin{array}{c|c} \overline{A_1} & 0 \\ \hline \overline{B_{21}} & \overline{A_2} \end{array} \right]. \tag{3.6}$$

And, $\left[\overline{B_{21}} \ \overline{A_2} \right]$ extract from Eq. (3.6). The number of the positive factor of the ith line of this matrix is set to n_{2i}, and the matrix which divided each factor of the ith line by n_{2i} is set to $\left[\widehat{B_{21}} \ \widehat{A_2} \right]$. Thereby, we can calculate $\hat{b_2}$ as follows:

$$\widehat{b_2} = \widehat{B_{21}} u_1. \tag{3.7}$$

When cluster 1 consists of one element, it is estimated that it is $u_1 = 1$. $\widehat{b_2}$ is an evaluation value given from cluster 1 to cluster 2.

The parameters p_i representing the level of importance of each component for entire system reliability can be estimated by using u_2 expressed with Eq. (3.8) from the above mentioned results:

$$\widehat{b_2} + \widehat{A_2} u_2 = u_2. \tag{3.8}$$

3.2 Reliability Assessment for Entire System

3.2.1 Inflection S-Shaped SRGM

We apply the existing inflection S-shaped SRGM for reliability assessment entire system. Thus, we consider the following structure of mean value function $S(t)$ because an NHPP model is characterized by its mean value function:

$$S(t) = \frac{a(1 - e^{-bt})}{1 + C \cdot e^{-bt}} \quad (a > 0, b > 0, C > 0), \tag{3.9}$$

where a is the expected number of initial inherent faults, and b the software failure rate per inherent faults. Moreover, we assume that C represents the following weighted

average in term of weight parameter p_i estimated by ANP and inflection rate c_i in Eq. (3.10):

$$C = \frac{\sum_{i=1}^{n} p_i \cdot c_i}{\sum_{i=1}^{n} p_i} = \sum_{i=1}^{n} p_i \cdot c_i, \tag{3.10}$$

where n represents the number of software component, p_i the weight parameter for each component, and c_i the inflection rate for the ith software component.

3.2.2 Software Reliability Assessment Measures

We can give the following expressions as software reliability assessment measures derived from NHPP model given Eq. (3.10):

- The expected number of remaining faults
  The expected number of faults remaining in the system at testing-time t, which is obtained as the expectation of random variable $\{N(\infty) - N(t)\}$, is given as follow:

$$N_c(t) \equiv E[N(\infty) - N(t)] = a - S(t). \tag{3.11}$$

- Software reliability
  The software reliability can be defined as the probability that a software failure does not occur during the time-interval $(t, t + x](t \geq 0, x \geq 0)$ after testing-time t. The software reliability is given by

$$R_c(x|t) = \exp[S(t) - S(t + x)] \quad (t \geq 0, x \geq 0). \tag{3.12}$$

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. E. Kinoshita, *Introductory AHP (in Japanese)* (JUSE Press, Tokyo, 2000)
3. E. Kinoshita, *Theory of AHP and Its Application* (JUSE Press, Tokyo, 2000)

Chapter 4
Stochastic Differential Equation Models for OSS Reliability Analysis

4.1 Introduction

Network technologies have made rapid progress with the dissemination of computer systems in all areas. These network technologies become increasingly more complex in a wide sphere. The mainstream of software development environment is the development paradigms such as concurrent distributed development environment and the so-called open source project by using network computing technologies. Especially, an OSS (open source software) system is used all over the world. Such OSS systems which serve as key components of critical infrastructures in our society are still ever-expanding now. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting the distributed development model in such open source projects includes GNU/Linux operating system,[1] Apache Web server, and so on.[2] However, the poor handling of the quality and customer support prohibit the progress of OSS. We focus on the problems in the software quality, that prohibit the progress of OSS.

Especially, software reliability growth models (SRGM's) [1] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for new distributed development paradigm as typified by the open source project has only a few presented [2–5]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for OSS, it is necessary to grasp the situation of registration for bug tracking system, degree of maturation of OSS, and so on.

In this chapter, we focus on an OSS developed under the open source project. We discuss a useful software reliability assessment method in open source project as a typical case of new distributed development paradigm. Especially, we propose

[1] Linux is a Registered Trademark of Linus Torvalds.

[2] Other company, product, or service names may be trademarks or service marks of others.

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_4

a software reliability growth model based on stochastic differential equations in order to consider the active state of the open source project. Especially, we assume that the software failure intensity depends on the time, and the software fault-report phenomena on the bug tracking system keep an irregular state. Also, we analyze actual software fault-count data to show numerical examples of software reliability assessment for the OSS. Moreover, we compare our model with the conventional model based on stochastic differential equations in terms of goodness-of-fit for actual data. We show that the proposed model can assist improvement of quality for OSS systems developed under the open source project.

4.2 Stochastic Differential Equation Modeling

Let $S(t)$ be the number of faults in the OSS system at testing time t ($t \geq 0$). Suppose that $S(t)$ takes on continuous real values. Since latent faults in the OSS system are detected and eliminated during the operational phase, $S(t)$ gradually increases as the operational procedures go on. Thus, under common assumptions for software reliability growth modeling, we consider the following linear differential equation:

$$\frac{dS(t)}{dt} = \lambda(t)S(t), \tag{4.1}$$

where $\lambda(t)$ is the intensity of inherent software failures at operational time t and is a non-negative function.

In most cases, the faults of OSS are not reported to the bug tracking system at the same time as fault detect but rather reported to the bug tracking system with the time lag of fault detection and report. As for the fault report to the bug tracking system, we consider that the software fault-report phenomena on the bug tracking system keep an irregular state. Moreover, the addition and deletion of software component is repeated under the development of OSS, i.e., we consider that the software failure intensity depends on the time.

Therefore, we suppose that $\lambda(t)$ in Eq. (4.1) has the irregular fluctuation. That is, we extend Eq. (4.1) to the following stochastic differential equation [6, 7]:

$$\frac{dS(t)}{dt} = \{\lambda(t) + \sigma\gamma(t)\}\, S(t), \tag{4.2}$$

where σ is a positive constant representing a magnitude of the irregular fluctuation and $\gamma(t)$ a standardized Gaussian white noise.

We extend Eq. (4.2) to the following stochastic differential equation of an Itô type:

$$dS(t) = \left\{\lambda(t) + \frac{1}{2}\sigma^2\right\} S(t)dt + \sigma S(t)dW(t), \tag{4.3}$$

where $W(t)$ is a one-dimensional Wiener process which is formally defined as an integration of the white noise $\gamma(t)$ with respect to time t. The Wiener process is a Gaussian process and it has the following properties:

$$\Pr[W(0) = 0] = 1, \tag{4.4}$$

$$\mathrm{E}[W(t)] = 0, \tag{4.5}$$

$$\mathrm{E}[W(t)W(t')] = \mathrm{Min}[t, t']. \tag{4.6}$$

By using Itô's formula [6, 7], we can obtain the solution of Eq. (4.2) under the initial condition $S(0) = v$ as follows [8]:

$$S(t) = v \cdot \exp\left(\int_0^t \lambda(s)ds + \sigma W(t)\right), \tag{4.7}$$

where v is the number of detected faults for the previous software version. Using solution process $S(t)$ in Eq. (4.7), we can derive several software reliability measures.

Moreover, we define the intensity of inherent software failures, $\lambda(t)$, as follows:

$$\int_0^t \lambda(s)ds = (1 - \exp[-\alpha t]), \tag{4.8}$$

where α is an acceleration parameter of the intensity of inherent software failures.

From Eq. (4.7), we can confirm that the number of detected faults can not converge to a finite value as the following equation.

$$\lim_{t\to\infty} S(t) = \infty. \tag{4.9}$$

The operating environment of OSS has the characteristics of the susceptible to various operational environments. Therefore, it is different from conventional software systems developed under the identical organization. Then, the expected number of detected faults continues to increase from the effect of the interaction among various operational environments, i.e., the number of detected faults can not converge to a finite value [9–11].

4.3 Method of Maximum-Likelihood

In this section, the estimation method of unknown parameters α and σ in Eq. (4.7) is presented. Let us denote the joint probability distribution function of the process $S(t)$ as

$$\begin{aligned} &P(t_1,\ y_1;\ t_2,\ y_2; \ldots; t_K,\ y_K) \\ &\quad \equiv \Pr[N(t_1) \le y_1, \ldots, N(t_K) \le y_K | S(t_0) = v], \end{aligned} \tag{4.10}$$

where $S(t)$ is the cumulative number of faults detected up to the operational time t ($t \geq 0$), and denote its density as

$$p(t_1,\ y_1;\ t_2,\ y_2; \ldots; t_K,\ y_K) \equiv \frac{\partial^K P(t_1,\ y_1;\ t_2,\ y_2; \ldots; t_K,\ y_K)}{\partial y_1 \partial y_2 \ldots \partial y_K}. \tag{4.11}$$

Since $S(t)$ takes on continuous values, we construct the likelihood function l for the observed data $(t_k, y_k)(k = 1, 2, \ldots, K)$ as follows:

$$l = p(t_1,\ y_1;\ t_2,\ y_2; \ldots; t_K,\ y_K). \tag{4.12}$$

For convenience in mathematical manipulations, we use the following logarithmic likelihood function:

$$L = \log l. \tag{4.13}$$

The maximum-likelihood estimates α^* and σ^* are the values making L in Eq. (4.13) maximize. These can be obtained as the solutions of the following simultaneous likelihood equations [8]:

$$\frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial \sigma} = 0. \tag{4.14}$$

4.4 Software Reliability Assessment Measures

4.4.1 Expected Numbers of Detected Faults and Their Variances

We consider the mean number of faults detected up to operational time t. The density function of $W(t)$ is given by:

$$f(W(t)) = \frac{1}{\sqrt{2\pi t}} \exp\left\{-\frac{W(t)^2}{2t}\right\}. \tag{4.15}$$

Information on the current number of detected faults in the system is important to estimate the situation of the progress on the software operational procedures. Since it is a random variable in our model, its expected value and variance can be useful measures. We can calculate them from Eq. (4.7) as follows [8]:

$$\mathrm{E}[S(t)] = v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right), \tag{4.16}$$

$$\begin{aligned}\mathrm{Var}[S(t)] &= \mathrm{E}[\{S(t) - \mathrm{E}[S(t)]\}^2] \\ &= v^2 \cdot \exp\left(2\int_0^t \lambda(s)ds + \sigma^2 t\right) \\ &\quad \cdot \left\{\exp(\sigma^2 t) - 1\right\},\end{aligned} \tag{4.17}$$

where $\mathrm{E}[S(t)]$ is the expected number of faults detected up to time t.

4.4.2 Mean Time Between Software Failures

The instantaneous mean time between software failures (denoted by $\mathrm{MTBF_I}$) is useful to measure the property of the frequency of software failure-occurrence.

Instantaneous MTBF is approximately given by:

$$MTBF_I(t) = \frac{1}{\mathrm{E}\left[\frac{dS(t)}{dt}\right]}. \tag{4.18}$$

Therefore, we have the following instantaneous MTBF.

$$\begin{aligned}MTBF_I(t) = 1 \Big/ \Bigg\{ & v\left(\lambda(t) + \frac{1}{2}\sigma^2\right) \\ & \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right)\Bigg\}.\end{aligned} \tag{4.19}$$

Also, Cumulative MTBF is approximately given by:

$$MTBF_C(t) = \frac{t}{\mathrm{E}[S(t)]}. \tag{4.20}$$

Therefore, we have the following cumulative mean time between software failures (denoted by $\mathrm{MTBF_C}$).

$$MTBF_C(t) = \frac{t}{v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right)}. \tag{4.21}$$

4.4.3 Coefficient of Variation

Also, we can derive the following coefficient of variation from Eq. (4.7):

$$CV(t) \equiv \frac{\sqrt{\mathrm{Var}[S(t)]}}{\mathrm{E}[S(t)]}. \tag{4.22}$$

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. A. MacCormack, J. Rusnak, C.Y. Baldwin, Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Inf. J. Manage. Sci. **52**(7), 1015–1030 (2006)
3. G. Kuk, Strategic interaction and knowledge sharing in the KDE developer mailing list. Inf. J. Manage. Sci. **52**(7), 1031–1042 (2006)
4. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. Proc. Workshop Open Source Softw. Eng. (WOSSE) **30**(4), 67–72 (2005)
5. P. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)* (2004), pp. 263–272
6. L. Arnold, *Stochastic Differential Equations-Theory and Applications* (Wiley, New York, 1974)
7. E. Wong, *Stochastic Processes in Information and Systems* (McGraw-Hill, New York, 1971)
8. S. Yamada, M. Kimura, H. Tanaka, S. Osaki, Software reliability measurement and assessment with stochastic differential equations. IEICE Trans. Fundam. **E77-A**(1), 109–116 (1994)
9. Y. Tamura, S. Yamada, Comparison of software reliability assessment methods for open source software, in *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005)–Volume II*, Fukuoka, Japan, 20–22 July 2005, pp. 488–492
10. Y. Tamura, S. Yamada, A method of user-oriented reliability assessment for open source software and its applications, in *Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, 8–11 October 2006, pp. 2185–2190
11. Y. Tamura, S. Yamada, Optimisation analysis for reliability assessment based on stochastic differential equation modelling for open source software. Int. J. Syst. Sci. **40**(4), 429–438 (2009). Taylor & Francis, United Kingdom

Chapter 5
Hazard Rates for Embedded OSS Reliability Analysis

5.1 Introduction

Network technologies have made rapid progress with the dissemination of computer systems in all areas. These network technologies become increasingly more complex in a wide sphere. Then, software development environment has been changing into new development paradigms such as concurrent distributed development environment and the so-called open source project by using network computing technologies. The case of success includes OSS (Open Source Software) systems which serve as key components of critical infrastructures in our society. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting such open source projects includes Apache HTTP server [1], Firefox Web browser [2], and GNU/Linux operating system. However, the poor handling of quality problem and customer support has limited the progress of OSS. Because the development cycle of OSS has no testing-phase. We focus on software quality/reliability problems that can prohibit the progress of embedded OSS.

In particular, software reliability growth models (SRGM's) [3–5] and hazard rate models [6–9] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for new distributed development paradigms as typified by the open source project has only a few presented [10–12]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for OSS, it is necessary to grasp the situation of registration for bug tracking system, the degree of maturation of OSS, and so on. In particular, an embedded OSS known as one of OSS's has been gaining a lot of attention in the embedded system area, i.e., Android, BusyBox, TRON, etc. However, the poor handling of quality problem and customer support has limited the progress of embedded OSS. Also, it is difficult for developers to assess the reliability and portability of embedded OSS on a single-board computer. The term

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_5

"porting-phase" means the rebuilding process in which the developers create an OS/application developed for the specific computer system to suit another computer system. From above mentioned problems, many companies have been hesitant to innovate the embedded OSS.

Many fault-counting type SRGM's have been applied to assess the reliability for quality management and testing-progress control of software development. However, it is difficult to apply the SRGM's to the OSS, because the number of detected faults in the OSS project can not converge to a finite value [13, 14]. In fact, there are several SRGM's that can be applied in the situation discussed above, i.e., the Weibull and Log-logistic SRGM's, and so on [3]. In particular, in case that the number of detected faults can not converge to a finite value, it is difficult to assess whether the porting phase will succeed by using reliability assessment measures derived from SRGM's. Also, the hazard rate model's have the simple structure.

As another more challenging aspect of the embedded OSS project, the embedded OSS includes several software components in terms of hardware such as a device driver. An application OSS features that the number of detected faults can not converge to a finite value. Therefore, we can apply SRGM's based on such assumption. On the other hand, the characteristics of embedded OSS include that the number of detected faults can not converge to a finite value. However, it is difficult to apply the conventional SRGM's to embedded OSS, because the embedded OSS includes several software components in terms of hardware such as a device driver.

For above mentioned reason, it is difficult to apply the conventional infinite failure SRGM's to embedded OSS. Therefore, we apply the SRGM's based on a software failure hazard rate (abbreviated as the *hazard rate model*) in place of the fault-counting type SRGM to the embedded OSS.

In this chapter, we propose a method of software reliability assessment based on a flexible hazard rate model for embedded OSS. Also, we derive several assessment measures from the model. In particular, we show several numerical results of reliability assessment for our hazard rate model. Moreover, we compare the goodness-of-fit of our model discussed in this chapter with the conventional hazard rate models. Furthermore, we discuss the optimal software release problem for the porting-phase based on the total expected software maintenance cost.

5.2 Flexible Hazard Rate Model for Embedded OSS

In this chapter, we assume that the software failures occurring in the porting-phase of embedded OSS are categorized in the following types:

A1. the software failure caused by the latent fault in the embedded OSS,
A2. the software failure caused by the latent fault in the specific software components (i.e., device driver).

In the assumption above, A1 is selected by probability p and A2 selected by probability $(1-p)$. Also, we can not distinguish between A1 and A2 software failures.

The time interval between successive faults of $(k-1)$th and kth is represented as the random variable X_k $(k = 1, 2, \ldots)$. Therefore, we can define the hazard rate function $z_k(x)$ for X_k as follows [15, 16]:

$$z_k(x) = p \cdot z_k^1(x) + (1-p) \cdot z_k^2(x) \quad (k = 1,\ 2,\ \ldots;\ 0 \le p \le 1), \tag{5.1}$$

$$z_k^1(x) = D(1 - \alpha \cdot e^{-\alpha k})^{k-1} \quad (k = 1,\ 2,\ \ldots;\ -1 < \alpha < 1,\ D > 0), \tag{5.2}$$

$$z_k^2(x) = \phi\{N - (k-1)\} \quad (k = 1,\ 2,\ \ldots,\ N;\ N > 0,\ \phi > 0), \tag{5.3}$$

where the notations in Eqs. (5.1)–(5.3) are represented as follows:

$z_k^1(x)$	the hazard rate for the A1 software failure,
α	the shape parameter representing the active state of OSS project,
D	the initial hazard rate for the 1st software failure,
$z_k^2(x)$	the hazard rate for the A2 software failure,
N	the number of latent faults in the specific software components,
ϕ	the hazard rate per inherent fault,
p	the weight parameter for $z_k^1(x)$.

Equation (5.2) means the hazard rate for a software failure-occurrence phenomenon for the embedded OSS. On the other hand, Eq. (5.3) represents the hazard rate for a software failure-occurrence phenomenon for the specific software components. Thus, our model simultaneously describes both the software failure occurring at the embedded OSS installed to embedded system by Eq. (5.2) and the software failure occurring at the specific software components such as the device driver.

In particular, our model includes both the modified Moranda model [8] and the conventional Jelinski–Moranda(J-M) model [7]. Equation (5.2) based on the Moranda model means that the initial hazard rate for the 1st software failure geometrically decreases with the active state of OSS. Also, we assume that the active state of OSS grows exponentially.

5.3 Reliability Assessment Measures

In porting-phase of embedded OSS, the distribution function of $X_k (k = 1,\ 2,\ \ldots)$ representing the time-interval between successive faults of $(k-1)$th and kth is defined as:

$$F_k(x) \equiv \Pr\{X_k \le x\} \quad (x \ge 0), \tag{5.4}$$

where $\Pr\{A\}$ represents the occurrence probability event A. Therefore, the following derived function means the probability density function of X_k:

$$f_k(x) \equiv \frac{dF_k(x)}{dx}. \tag{5.5}$$

Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(0, x]$ after the porting-phase. The software reliability is given by

$$R_k(x) \equiv \Pr\{X_k > x\} = 1 - F_k(x). \tag{5.6}$$

From Eqs. (5.4) and (5.5), the hazard rate is given by the following equation:

$$z_k(x) \equiv \frac{f_k(x)}{1 - F_k(x)} = \frac{f_k(x)}{R_k(x)}, \tag{5.7}$$

where the hazard rate means the software failure rate after the porting-phase when the software failure does not occur during the time-interval $(0, x]$.

Therefore, we can obtain the software reliability assessment measures from our hazard rate model in Eq. (5.1). The probability density function can be derived as

$$\begin{aligned} f_k(x) = {} & \left\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} + (1-p)\phi(N-k+1)\right\} \\ & \cdot \exp\left[-\left\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \right.\right. \\ & \left.\left. + (1-p)\phi(N-k+1)\right\}\cdot x\right]. \end{aligned} \tag{5.8}$$

Also, the software reliability represents the following equation:

$$\begin{aligned} R_k(x) = \exp\Bigg[& -\left\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \right. \\ & \left. + (1-p)\phi(N-k+1)\right\}\cdot x\Bigg]. \end{aligned} \tag{5.9}$$

Moreover, the mean time between software failures(MTBF) is given as follows:

$$\begin{aligned} \mathrm{E}[X_k] = 1 \Big/ & \left\{pD(1-\alpha\cdot e^{-\alpha k})^{k-1} \right. \\ & \left. + (1-p)\phi(N-k+1)\right\}. \end{aligned} \tag{5.10}$$

References

1. The Apache HTTP Server Project, The Apache Software Foundation, http://httpd.apache.org/
2. Mozilla.org, Mozilla Foundation, http://www.mozilla.org/
3. M.R. Lyu (ed.), *Handbook of Software Reliability Engineering* (IEEE Computer Society Press, Los Alamitos, 1996)

4. J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, 1987)
5. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
6. G.J. Schick, R.W. Wolverton, An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **SE-4**(2), 104–120 (1978)
7. Z. Jelinski, P.B. Moranda, Software reliability research, in *Statistical Computer Performance Evaluation*, ed. by W. Freiberger (Academic Press, New York, 1972), pp. 465–484
8. P.B. Moranda, Event–altered rate models for general reliability analysis. IEEE Trans. Reliab. **R-28**(5), 376–381 (1979)
9. M. Xie, On a Generalization of the J-M model. Proc. Reliab. '89 **5**, Ba/3/1–5 Ba/3/7 (1989)
10. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. Proc. Workshop Open Source Softw. Eng. (WOSSE) **30**(4), 67–72 (2005)
11. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering* (*FSE-12*) (2004), pp. 263–272
12. J. Norris, Mission-critical development with open source software. IEEE Softw. Mag. **21**(1), 42–49 (2004)
13. Y. Tamura, S. Yamada, Comparison of software reliability assessment methods for open source software, in *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems* (*ICPADS2005*), vol. II, Fukuoka, Japan, 20–22 July 2005, pp. 488–492
14. Y. Tamura, S. Yamada, A method of user-oriented reliability assessment for open source software and its applications, *Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, 8–11 October 2006, pp. 2185–2190
15. Y. Tamura, S. Yamada, Reliability assessment based on hazard rate model for an embedded OSS porting phase. J. Softw. Test. Verif. Reliab. **23**(1), 77–88 (2013). doi:10.1002/stvr.455. (Article first published online: 17 March, 2011), Wiley
16. Y. Tamura, S. Yamada, Performability analysis considering debugging behaviors for open source solution. Int. J. Perform. Eng. **9**(1), 13–21 (2013)

Chapter 6
Reliability Analysis for Open Source Solution

6.1 Introduction

At present, there is growing interest in the next-generation software development paradigm by using network computing technologies such as a cloud computing. Considering the software development environment, one has been changing into new development paradigms such as concurrent distributed development environment and the so-called open source project by using network computing technologies.

The successful experience of adopting the distributed development model in such open source projects includes GNU/Linux operating system, Apache HTTP server, and so on. However, the poor handling of the quality and customer support prohibits the progress of OSS. We focus on the problems of software quality, which prohibit the progress of OSS. Especially, a large-scale open source solution is now attracting attention as the next-generation software development paradigm. Also, the large-scale open source solution is composed of several OSS's.

Many software reliability growth models (SRGM's) [1] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for new distributed development paradigm as typified by the open source project has only a few presented [2–5]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for open source solution, it is necessary to grasp the situation of registration for bug tracking system, the combination status of OSS's, the degree of maturation of OSS, and so on.

In this chapter, we focus on an open source solution developed under several OSS's. We discuss a useful method of software reliability assessment in open source solution as a typical case of next-generation distributed development paradigm. Then, we propose a software reliability growth model based on stochastic differential equations in order to consider the active state of the open source project and the component collision of OSS. Then, we assume that the software failure intensity depends on the time, and the software fault-report phenomena on the bug tracking system keep an

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_6

irregular state. Also, we analyze actual software fault-count data to show numerical examples of software reliability assessment for the open source solution. Especially, we derive several reliability assessment measures from our model. Then, we show that the proposed model can assist improvement of quality for open source solution developed under several OSS.

6.2 Stochastic Differential Equation Model

Let $S(t)$ be the number of detected faults in the open source solution by testing time t ($t \geq 0$). Suppose that $S(t)$ takes on continuous real values. Since latent faults in the open source solution are detected and eliminated during the testing-phase, $S(t)$ gradually increases as the testing procedures go on. Thus, under common assumptions for software reliability growth modeling, we consider the following linear differential equation:

$$\frac{dS(t)}{dt} = \lambda(t)S(t), \tag{6.1}$$

where $\lambda(t)$ is the intensity of inherent software failures at testing time t and is a non-negative function.

Generally, it is difficult for users to use all functions in open source solution, because the connection state among open source components is unstable in the testing-phase of open source solution. Considering the characteristic of open source solution, the software fault-report phenomena keep an irregular state in the early stage of testing-phase. Moreover, the addition and deletion of software components are repeated under the development of an OSS system, i.e., we consider that the software failure intensity depends on the time.

Therefore, we suppose that $\lambda(t)$ in Eq. (6.1) has the irregular fluctuation. That is, we extend Eq. (6.1) to the following stochastic differential equation [6, 7]:

$$\frac{dS(t)}{dt} = \{\lambda(t) + \sigma\mu(t)\gamma(t)\}\, S(t), \tag{6.2}$$

where σ is a positive constant representing a magnitude of the irregular fluctuation, $\gamma(t)$ a standardized Gaussian white noise, and $\mu(t)$ the collision level function of open source component.

We extend Eq. (6.2) to the following stochastic differential equation of an Itô type:

$$\begin{aligned} dS(t) = &\left\{\lambda(t) + \frac{1}{2}\sigma^2\mu(t)^2\right\} S(t)dt \\ &+ \sigma\mu(t)S(t)dW(t), \end{aligned} \tag{6.3}$$

where $W(t)$ is a one-dimensional Wiener process which is formally defined as an integration of the white noise $\gamma(t)$ with respect to time t. The Wiener process is a Gaussian process and it has the following properties:

$$\Pr[W(0) = 0] = 1, \tag{6.4}$$

$$\mathrm{E}[W(t)] = 0, \tag{6.5}$$

$$\mathrm{E}[W(t)W(t')] = \mathrm{Min}[t, t']. \tag{6.6}$$

By using Itô's formula [6, 7], we can obtain the solution of Eq. (6.3) under the initial condition $S(0) = v$ as follows [8]:

$$S(t) = v \cdot \exp\left(\int_0^t \lambda(s)ds + \sigma\mu(t)W(t) \right), \tag{6.7}$$

where v is the total number of detected faults for the applied OSS's. Using solution process $S(t)$ in Eq. (6.7), we can derive several software reliability measures.

Moreover, we define the intensity of inherent software failures in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$, and the collision level function $\mu(t)$ as follows:

$$\int_0^t \lambda_1(s)ds = (1 - \exp[-\alpha t]), \tag{6.8}$$

$$\int_0^t \lambda_2(s)ds = (1 - (1 + \alpha t)\exp[-\alpha t]), \tag{6.9}$$

$$\mu(t) = \exp[-\beta t], \tag{6.10}$$

where α is an acceleration parameter of the intensity of inherent software failures, and β the growth parameter of the stability of open source solution.

6.3 Method of Maximum-Likelihood

In this section, the estimation method of unknown parameters α, β and σ in Eq. (6.7) is presented. Let us denote the joint probability distribution function of the process $S(t)$ as

$$\begin{aligned} &P(t_1,\ y_1;\ t_2,\ y_2; \ldots;\ t_K,\ y_K) \\ &\quad \equiv \Pr[N(t_1) \le y_1, \ldots,\ N(t_K) \le y_K \\ &\quad |S(t_0) = v], \end{aligned} \tag{6.11}$$

where $S(t)$ is the cumulative number of faults detected up to the testing time t ($t \ge 0$), and denote its density as

$$\begin{aligned} &p(t_1,\ y_1;\ t_2,\ y_2; \ldots;\ t_K,\ y_K) \\ &\quad \equiv \frac{\partial^K P(t_1,\ y_1;\ t_2,\ y_2; \ldots;\ t_K,\ y_K)}{\partial y_1 \partial y_2 \ldots \partial y_K}. \end{aligned} \tag{6.12}$$

Since $S(t)$ takes on continuous values, we construct the likelihood function l for the observed data $(t_k, y_k)(k = 1, 2, \ldots, K)$ as follows:

$$l = p(t_1,\ y_1;\ t_2,\ y_2; \ldots;\ t_K,\ y_K). \tag{6.13}$$

For convenience in mathematical manipulations, we use the following logarithmic likelihood function:

$$L = \log l. \tag{6.14}$$

The maximum-likelihood estimates α^*, β^* and σ^* are the values making L in Eq. (6.14) maximize. These can be obtained as the solutions of the following simultaneous likelihood equations [8]:

$$\frac{\partial L}{\partial \alpha} = \frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial \sigma} = 0. \tag{6.15}$$

6.4 Software Reliability Assessment Measures

6.4.1 Expected Numbers of Detected Faults

We consider the expected number of faults detected up to testing time t. The density function of $W(t)$ is given by:

$$f(W(t)) = \frac{1}{\sqrt{2\pi t}} \exp\left\{-\frac{W(t)^2}{2t}\right\}. \tag{6.16}$$

Information on the cumulative number of detected faults in the system is important to estimate the situation of the progress on the software testing procedures. Since $S(t)$ is a random variable in our model, its expected value can be a useful measure. We can calculate them from Eq. (6.7) as follows [8]:

$$\mathrm{E}[S(t)] = v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2\mu(t)^2}{2}t\right), \tag{6.17}$$

where $\mathrm{E}[S(t)]$ is the expected number of faults detected up to time t. Also, the expected number of remaining faults at time t can obtain as follows:

$$\mathrm{E}[N(t)] = v \cdot e - \mathrm{E}[S(t)], \tag{6.18}$$

where v is the total number of detected faults for the applied OSS's, and e means exp(1).

6.4.2 Mean Time Between Software Failures

The instantaneous mean time between software failures (denoted by $\mathrm{MTBF_I}$) is useful to measure the property of the frequency of software failure-occurrence. Then, the instantaneous MTBF is approximately given by:

$$MTBF_I(t) = \frac{1}{\mathrm{E}\left[\frac{dS(t)}{dt}\right]}. \tag{6.19}$$

Also, the cumulative MTBF is approximately given by:

$$MTBF_C(t) = \frac{t}{\mathrm{E}[S(t)]}. \tag{6.20}$$

6.4.3 Coefficient of Variation

Also, we can use the coefficient of variation as the measure of variation without the effect of mean value. We can derive the following coefficient of variation from Eq. (6.7):

$$CV(t) \equiv \frac{\sqrt{\mathrm{Var}[S(t)]}}{\mathrm{E}[S(t)]}. \tag{6.21}$$

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. A. MacCormack, J. Rusnak, C.Y. Baldwin, Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Inf. J. Manag. Sci. **52**(7), 1015–1030 (2006)
3. G. Kuk, Strategic interaction and knowledge sharing in the KDE developer mailing list. Inf. J. Manag. Sci. **52**(7), 1031–1042 (2006)
4. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. Proc. Workshop Open Source Softw. Eng. (WOSSE) **30**(4), 67–72 (2005)
5. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems. in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)*, pp. 263–272, 2004
6. L. Arnold, *Stochastic Differential Equations-Theory and Applications* (Wiley, New York, 1974)
7. E. Wong, *Stochastic Processes in Information and Systems* (McGraw-Hill, New York, 1971)
8. S. Yamada, M. Kimura, H. Tanaka, S. Osaki, Software reliability measurement and assessment with stochastic differential equations. IEICE Trans. Fundam. **E77-A**(1), 109–116 (1994)

Chapter 7
Reliability Analysis for Mobile OSS

7.1 Introduction

At present, OSS (Open Source Software) systems serve as key components of critical infrastructures in the society. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting such open source projects includes Apache HTTP server, MySQL database server, OpenStack cloud software, Firefox Web browser, and GNU/Linux operating system, etc. However, the poor handling of quality problem and customer support has limited the progress of OSS, because the development cycle of OSS has no testing-phase. We focus on software quality/reliability problems that they can prohibit the progress of embedded OSS. In particular, an embedded OSS known as one of OSS's has been gaining a lot of attention in the embedded system area, i.e., Android [1], BusyBox [2], Firefox OS [3], etc. However, the installer software developed under the third-party developers indirectly effect on the reliability in area of a mobile device. Also, it is difficult for developers to assess reliability and portability of the porting-phase in case of installing the embedded OSS on a single-board computer. The term "porting-phase" means the rebuilding process in which the developers create an OS/application developed for the specific computer system to suit another computer systems in terms of portability. From above mentioned problems, many companies have been hesitant to innovate the embedded OSS, because an OSS includes several software versions.

Especially, software reliability growth models (SRGM's) [4–6] and the related hazard rate models [7–10] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for a new distributed development paradigm as typified by the open source project has only a few presented [11–13]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for the software developed under the third-party developers, it is necessary to grasp the situation of installer

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_7

software, the network traffic, the installed software, etc. Then, it is very important to consider the status of network traffic in terms of the reliability assessment in the following standpoint:

- In case of the open source, the weakness of reliability and security becomes a significant problem via a computer network.
- By using the installer software, the various third-party software are installed via the network.
- In case of the mobile device, the network access devices are frequently used by many software installed via the installer software.

Many fault-counting type SRGM's have been applied to assess the reliability for quality management and testing-progress control of software development. However, it is difficult to apply the SRGM's for assessing quality/reliability of the software developed under the third-party developers such as OSS. In other words, the testing-phase is nonexistent in the open source development paradigm. In fact, there are several SRGM's that can be applied in the above situation, i.e., the Weibull and Log-logistic SRGM's, and so on [4]. In particular, in case the software developed under the third-party developers, it is difficult to assess both the software failure and network traffic affected by using the installer software. As another more challenging aspect of the embedded OSS project, the embedded OSS includes several software components in terms of hardware such as a device driver [14, 15].

In this chapter, we propose a method of software reliability/portability assessment based on a hazard rate model and neural network for the mobile device. Also, we derive several assessment measures. In particular, we analyze actual software failure-occurrence time data to show numerical examples of software reliability/portability assessment for the mobile device. Then, we show that our model may assist quality improvement for mobile device systems development. Furthermore, we investigate a useful software reliability assessment method for the actual mobile device system development.

7.2 Hazard Rate Model Depending on the Change of Network Traffic

The time-interval between successive software failures of $(k-1)$th and kth is represented as the random variable X_k $(k = 1, 2, \ldots)$. Therefore, we can define the hazard rate function $z_k(x)$ at time x during the porting-phase for X_k as follows:

$$z_k(x) = w_k(x)\{N - (k-1)\} \quad (k = 1, 2, \ldots, \alpha;\ \phi > 0), \tag{7.1}$$

$$w_k(x) = \phi e^{-p_k} \quad (N > 0,\ \phi > 0,\ -1 < p_k < 1), \tag{7.2}$$

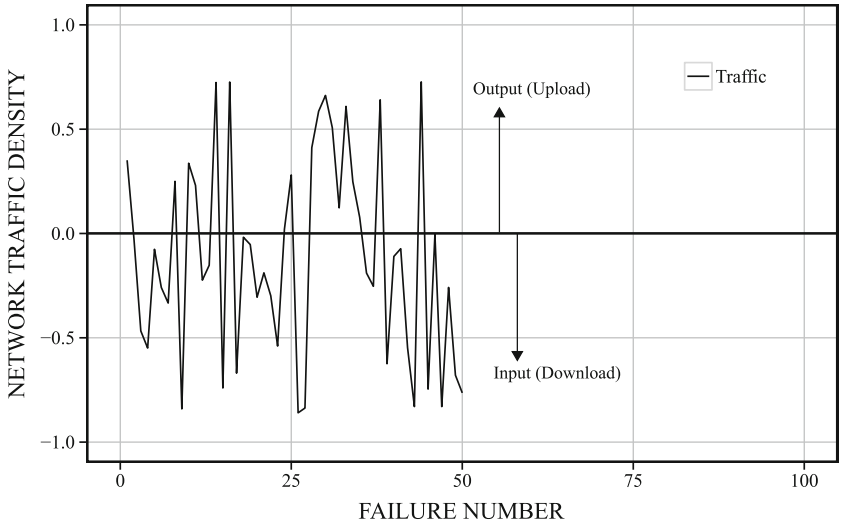

Fig. 7.1 The basic concept of network traffic density

where we can define the each parameter as follows:

$z_k(x)$ the hazard rate for the whole embedded software,
$w_k(x)$ the hazard rate per inherent fault considering the network traffic density,
N the number of latent faults,
ϕ the hazard rate per inherent fault,
p_k the changing rate of the network traffic density.

Equation (7.1) means the hazard rate for a software failure-occurrence phenomenon for the embedded software. Also, we assume that the hazard rate per inherent fault exponentially depends on the network traffic density in terms of the number of software failures k as shown in Fig. 7.1. In particular, the network traffic changes with the number of installed software and the addition of device drivers. Our model can describes the fault-detection phenomenon detected at among the installed software and device driver softwares installed to embedded system by the changing rate of network traffic density p_k included in Eq. (7.2). Thus, the embedded system shows a reliability regression trend if p_k is negative value. On the other hand, the embedded system shows a reliability growth trend if p_k is positive value.

7.3 Reliability Assessment Measures

In porting-phase of embedded OSS, the distribution function of $X_k(k = 1, 2, \ldots)$ representing the time-interval between successive software failures of $(k-1)$th and kth is defined as:

$$F_k(x) \equiv \Pr\{X_k \leq x\} \quad (x \geq 0), \tag{7.3}$$

where Pr{A} represents the occurrence probability of event A. Therefore, the following function means the probability density function of X_k:

$$f_k(x) \equiv \frac{dF_k(x)}{dx}. \tag{7.4}$$

Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(0, x]$ after the porting-phase. The software reliability is given by

$$R_k(x) \equiv \Pr\{X_k > x\} = 1 - F_k(x). \tag{7.5}$$

From Eqs. (7.3) and (7.4), the hazard rate is given by the following equation:

$$z_k(x) \equiv \frac{f_k(x)}{1 - F_k(x)} = \frac{f_k(x)}{R_k(x)}, \tag{7.6}$$

where the hazard rate means the software failure rate after the porting-phase when the software failure does not occur during the time-interval $(0, x]$.

Therefore, we can obtain the software reliability assessment measures from our hazard rate model represented by Eq. (7.1). The probability density function can be derived as

$$f_k(x) = \left\{\phi e^{-p_k}(N - k + 1)\right\} \cdot \exp\left[-\left\{\phi e^{-p_k}(N - k + 1)\right\} \cdot x\right]. \tag{7.7}$$

Especially, we can give the following expressions as software reliability assessment measures derived from our hazard rate model:

- *MTBF*
  The mean time between software failures(MTBF) is useful to measure the property of the frequency of software failure-occurrence, and is given by

$$\mathrm{E}[X_k] = \frac{1}{\phi e^{-p_k}(N - k + 1)}. \tag{7.8}$$

- *Software reliability*
  Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(t, t + x]$ $(t \geq 0, x \geq 0)$ given that the debugging progress time of porting-phase is t. The software reliability is given as follows:

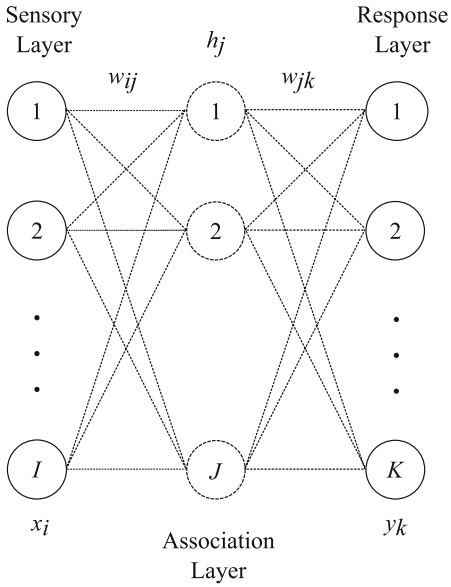

Fig. 7.2 The structure of 3-layered neural networks

$$R_k(x) = \exp\left[-\phi e^{-p_k} (N - k + 1) \cdot x \right]. \tag{7.9}$$

7.4 Estimation of Network Traffic Density

The structure of the neural networks applied in this chapter is shown in Fig. 7.2. In Fig. 7.2, $w_{ij}^1 (i = 1, 2, \ldots, I; j = 1, 2, \ldots, J)$ are the connection weights from ith unit on the sensory layer to jth unit on the association layer, and $w_{jk}^2 (j = 1, 2, \ldots, J; k = 1, 2, \ldots, K)$ denote the connection weights from jth unit on the association layer to kth unit on the response layer. Moreover, $x_i (i = 1, 2, \ldots, I)$ represent the normalized input values of ith unit on the sensory layer, and $y_k (k = 1, 2, \ldots, K)$ are the output values.

In Fig. 7.2, the input-output rules of each unit on each layer are given by

$$h_j = f\left(\sum_{i=1}^{I} w_{ij}^1 x_i \right), \tag{7.10}$$

$$y_k = f\left(\sum_{j=1}^{J} w_{jk}^2 h_j \right), \tag{7.11}$$

where a logistic activation function $f(\cdot)$ which is well-known as a sigmoid function given by the following equation:

$$f(x) = \frac{1}{1 + e^{-\theta x}}, \tag{7.12}$$

where θ is the slant of sigmoid function. We apply the multi-layered neural networks by back-propagation in order to learn the time series data in terms of the network traffic density [16]. We define the error function for Eq. (7.11) by the following equation:

$$E = \frac{1}{2} \sum_{k=1}^{K} (y_k - d_k)^2, \tag{7.13}$$

where $d_k (k = 1, 2, \ldots, K)$ are the target input values for the output values. We apply the normalized values of the time series data for network traffic density to the target input values $d_k (k = 1, 2, \ldots, K)$ for the output values, i.e., we consider the estimation and prediction model so that the property of the network traffic density accumulates on the connection weights of neural networks.

The rules for changing weights among units are shown as follows:

$$\begin{aligned} w_{jk}^2(r+1) &= w_{jk}^2(r) - \varepsilon \frac{\partial E}{\partial w_{jk}^2} \\ &= w_{jk}^2(r) \\ &\quad + \varepsilon (y_k - d_k) \\ &\quad \cdot f' \left(\sum_{j=1}^{J} w_{jk}^2(r) h_j \right) h_j, \end{aligned} \tag{7.14}$$

$$\begin{aligned} w_{ij}^1(r+1) &= w_{ij}^1(r) - \varepsilon \frac{\partial E}{\partial w_{ij}^1} \\ &= w_{ij}^1(r) \\ &\quad + \varepsilon \sum_{k=1}^{K} (p_k - d_k) f' \left(\sum_{j=1}^{J} w_{jk}^2(r) h_j \right) \\ &\quad \cdot w_{ij}^1(r) f' \left(\sum_{i=1}^{I} w_{ij}^1(r) x_i \right) x_i, \end{aligned} \tag{7.15}$$

where ε is a positive constant for learning in neural networks, and r represents the iteration of calculation symbolically, $w_{ij}^1(r+1)$ and $w_{jk}^2(r+1)$ are the new

approximate values of connection weights. $w_{ij}^1(r)$ and $w_{jk}^2(r)$ are also the prespecified values of connection weights. We estimate the new approximate values of connection weights $w_{ij}^1(r+1)$ and $w_{jk}^2(r+1)$ by Eqs. (7.14) and (7.15) repeatedly, until the approximate values satisfy the following convergence conditions:

$$\left| \frac{w_{jk}^2(r+1) - w_{jk}^2(r)}{w_{jk}^2(r)} \right| \leq \xi, \tag{7.16}$$

$$\left| \frac{w_{ij}^1(r+1) - w_{ij}^1(r)}{w_{ij}^1(r)} \right| \leq \xi. \tag{7.17}$$

From the above mentioned results, the estimated network traffic density is applied to the parameter p_k included in the proposed hazard rate model.

7.5 Parameter Estimation

Suppose that n data pairs $x_k (k = 1, 2, \ldots, n)$ are observed during the porting-phase, where $x^{(n)} = (x_1, x_2, \ldots, x_n)$ is the time-interval between successive $(k-1)$th and kth software failures. Then, the logarithmic likelihood function of the hazard rate model with $z_k(x)$ of Eq. (10.1) is given by

$$\begin{aligned} \mathrm{LLF}(N, \phi; x^n) = &\sum_{k=1}^{n} \log \left\{ \phi e^{-\mathrm{E}[p_k]} (N-k+1) \right\} \\ &- \left\{ \phi e^{-\mathrm{E}[p_k]} \sum_{k=1}^{n} (N-k+1) x_k \right\}. \end{aligned} \tag{7.18}$$

The maximum-likelihood estimates $\hat{N}$ and $\hat{\phi}$ for the unknown parameters N and ϕ can be obtained by solving the following simultaneous likelihood equations numerically:

$$\frac{\partial \mathrm{LLF}}{\partial N} = \frac{\partial \mathrm{LLF}}{\partial \phi} = 0. \tag{7.19}$$

In particular, we assume that $p_k (k = 1, 2, \ldots, n)$ are estimated by using neural network. Then, the mean value of the estimated $\widehat{p_k}$ is used in this parameter estimation. The procedures of reliability/portability analysis based on the proposed model for mobile device are shown as follows:

1. The embedded software developers process the data file in terms of the software failure-occurrence time data in the porting-phase of the embedded system for reliability/portability analysis.
2. The embedded software developers estimate the unknown parameters $p_k (k = 1, 2, \ldots, n)$ included in our hazard rate model by using the neural network.
3. The embedded software developers estimate the unknown parameters N and ϕ included in our hazard rate model by using the method of maximum-likelihood.
4. It is useful for embedded software developers to understand the debugging progress in porting-phase of embedded system development by using the MTBF in Eq. (7.8) and software reliability in Eq. (7.9) considering the change of network traffic as software reliability/portability assessment measures.

References

1. Open Handset Alliance, Android. http://www.android.com/
2. E. Andersen, BusyBox. http://www.busybox.net/
3. Firefox OS, Marketplace, Android–Partners–mozilla.org, Mozilla Foundation. http://www.mozilla.org/firefoxos/
4. M.R. Lyu (ed.), *Handbook of Software Reliability Engineering* (IEEE Computer Society Press, Los Alamitos, 1996)
5. J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, 1987)
6. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
7. G.J. Schick, R.W. Wolverton, An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **SE-4**(2), 104–120 (1978)
8. Z. Jelinski, P.B. Moranda, *Software Reliability Research, in Statistical Computer Performance Evaluation*, Freiberger (Academic Press, New York, 1972), pp. 465–484
9. P.B. Moranda, Event–altered Rate Models for General Reliability Analysis. IEEE Trans. Reliab. **R-28**(5), 376–381 (1979)
10. M. Xie, On a Generalization of the J-M Model, *Proceedings Reliability '89*, 5, Ba/3/1–5 Ba/3/7 (1989)
11. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach, in *Proceedings of the Workshop on Open Source Software Engineering (WOSSE)*, vol. 30, no. 4 (2005), pp. 67–72
12. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)* (2004), pp. 263–272
13. J. Norris, Mission-critical development with open source software. IEEE Softw. Mag. **21**(1), 42–49 (2004)
14. K.Y. Cai, D.B. Hu, C. Bai, H. Hu, T. Jing, Does software reliability growth behavior follow a non-homogeneous Poisson process. Inf. Softw. Technol. **50**(12), 1232–1247 (2008)
15. Y. Tamura, S. Yamada, Reliability assessment based on hazard rate model for an embedded OSS porting phase. J. Softw. Test. Verif. Reliab. **23**(1), 77–88 (2013)
16. E.D. Karnin, A simple procedure for pruning back-propagation trained neural networks. IEEE Trans. Neural Netw. **1**, 239–242 (1990)

Chapter 8
Reliability Analysis Tool for Embedded OSS

8.1 Introduction

Considering software development environment, there is growing interest in the next-generation software development paradigm by using network computing technologies such as an open source project and a cloud computing. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting such open source projects includes Apache HTTP server [1], Firefox Web browser [2], and GNU/Linux operating system.

Especially, software reliability growth models (SRGM's) [3–5] and the related hazard rate models [6–9] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, only a few effective methods of dynamic testing management for new distributed development paradigms as typified by the open source project have presented [10–12]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for Open Source Software (OSS), it is necessary to grasp the situation of registration for bug tracking system, the degree of maturity of OSS, and so on. Especially, an embedded OSS known as one of OSS's has been gaining a lot of attention in the embedded systems area, i.e., Android [13], BusyBox [14], TRON, etc. However, the poor handling of quality problem and customer support prohibits the progress of embedded OSS. Also, it is difficult for developers to assess reliability and portability of the porting-phase in case of installing the embedded OSS on a single-board computer. The term "porting-phase" means the rebuilding process in which the developers create an OS/application developed for the specific computer system to suit the another computer system. From above mentioned problems, many companies have been hesitant to introduce the embedded OSS.

Many fault-counting type SRGM's have been applied to assess the reliability for quality management and testing-progress control of software development. However, it is difficult to apply the SRGM's for assessing quality/reliability of the OSS,

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_8

because the number of detected faults in the OSS project can not converge to a finite value [15, 16]. In other words, the testing-phase is nonexistent in the open source development paradigm. In fact, there are several SRGM's that can be applied in the above situation, i.e., the Weibull and Log-logistic SRGM's, and so on [3]. Especially, in case that the number of detected faults can not converge to a finite value, it is difficult to assess whether the porting phase will succeed by using reliability assessment measures derived from SRGM's. As another more challenging aspect of the embedded OSS project, the embedded OSS includes several software components in terms of hardware such as device driver. Therefore, it is difficult to apply such conventional SRGM's to embedded OSS. Therefore, we apply the SRGM's based on a software failure hazard rate (abbreviated as the *hazard rate model*) in place of the fault-counting type SRGM to the embedded OSS.

In this chapter, we propose a method of software reliability assessment based on a flexible hazard rate model for embedded OSS. Also, we derive several assessment measures. Especially, we also develop the software reliability/portability assessment tool for the porting-phase of embedded system development by using Java programming language. Then, we analyze actual software failure-occurrence time-interval data to show numerical examples of software reliability/portability analysis for the porting-phase of embedded system development by using the proposed tool. Especially, we develop the tool considering optimal release problem based on a hazard rate model for the embedded open source software. Also, we add a new feature to our tool in order to compare our model with the existing models. Moreover, we show that the proposed tool can assist quality improvement for embedded OSS systems development.

8.2 Hazard Rate Model for Embedded OSS Porting Phase

In this chapter, we assume that the software faults detected at the porting-phase of embedded OSS include the following two types:

A1. the software failure caused by the latent fault of embedded OSS
A2. the software failure caused by the latent fault of unique software components (e.g., device driver).

In the assumption above, A1 is selected by probability p and A2 selected by probability $(1-p)$. Also, we can not distinguish between assumptions A1 and A2 in terms of the software faults. The time-interval between successive faults of $(k-1)$st and kth is represented as the random variable X_k $(k = 1, 2, \ldots)$. Therefore, we can define the hazard rate function $z_k(x)$ for X_k as follows:

$$z_k(x) = p \cdot z_k^1(x) + (1-p) \cdot z_k^2(x) \quad (k = 1,\ 2,\ \ldots;\ 0 \le p \le 1), \tag{8.1}$$

$$z_k^1(x) = D(1 - \alpha \cdot e^{-\alpha k})^{k-1} \quad (k = 1,\ 2,\ \ldots;\ -1 < \alpha < 1,\ D > 0), \tag{8.2}$$

$$z_k^2(x) = \phi\{N - (k-1)\} \quad (k = 1,\ 2,\ \ldots,\ N;\ N > 0,\ \phi > 0), \tag{8.3}$$

where each parameter is defined as follows:

$z_k^1(x)$	the hazard rate for the assumption A1,
α	the shape parameter representing the active state of OSS project,
D	the initial hazard rate for the 1st software failure,
$z_k^2(x)$	the hazard rate for the assumption A2,
N	the number of latent faults in unique software components,
ϕ	the hazard rate per inherent fault,
p	the weight parameter for $z_k^1(x)$.

Equation (8.2) means the hazard rate for a software failure-occurrence phenomenon for the embedded OSS. On the other hand, Eq. (8.3) represents the hazard rate for a software failure-occurrence phenomenon for the unique software components. Thus, our model defined by Eq. (8.1) simultaneously describes both the faults detected at the embedded OSS installed to embedded system by Eq. (8.2) and detected at the unique software components such as the device drivers.

In particular, our model includes both the modified Moranda model [8] and the conventional Jelinski–Moranda(J-M) model [7]. Equation (8.2) based on the Moranda model means that the initial hazard rate for the 1st software failure geometrically decreases with the active state of OSS. Also, we assume that the active state of OSS grows exponentially in accordance with the decreasing value of parameter α.

8.3 Reliability Assessment Measures

In porting-phase of embedded OSS, the distribution function of $X_k (k = 1, 2, \ldots)$ representing the time-interval between successive software failures of $(k-1)$st and kth is defined as:

$$F_k(x) \equiv \Pr\{X_k \leq x\} \quad (x \geq 0), \tag{8.4}$$

where Pr{A} represents the occurrence probability of event A. Therefore, the following function means the probability density function of X_k:

$$f_k(x) \equiv \frac{dF_k(x)}{dx}. \tag{8.5}$$

Therefore, we can obtain software reliability assessment measures from our hazard rate model represented by Eq. (10.1) as follows. The probability density function can be derived as

$$f_k(x) = \left\{ pD(1-\alpha \cdot e^{-\alpha k})^{k-1} + (1-p)\phi(N-k+1) \right\} \cdot \exp\left[- \left\{ pD(1-\alpha \cdot e^{-\alpha k})^{k-1} + (1-p)\phi(N-k+1) \right\} \cdot x \right]. \tag{8.6}$$

Also, the software reliability is given by the following equation:

$$R_k(x) = \exp\left[-\left\{ pD(1-\alpha \cdot e^{-\alpha k})^{k-1} + (1-p)\phi(N-k+1) \right\} \cdot x \right]. \quad (8.7)$$

Moreover, the mean time between software failures(MTBF) is given as follows:

$$\mathrm{E}[X_k] = \frac{1}{pD(1-\alpha \cdot e^{-\alpha k})^{k-1} + (1-p)\phi(N-k+1)}. \quad (8.8)$$

8.4 Optimal Release Problem for the Porting-Phase

We find the optimal release time of porting-phase by minimizing the total expected software cost in this section. Then, we discuss about the determination of optimal software release times minimizing the total expected software cost [17, 18]. We introduce the following cost parameters:

c_1 the testing cost per porting-time ($c_1 > 0$),
c_2 the fixing cost per fault during the porting-phase ($c_2 > 0$),
c_3 the fixing cost per fault after the release ($c_3 > c_2$).

Then, the expected software cost of OSS can be formulated as:

$$C_1(l) = c_1 \sum_{k=1}^{l} \mathrm{E}[X_k] + c_2 l, \quad (8.9)$$

where l is the number of software failure-occurrence.

Also, we can define the expected software cost for software components as follows:

$$C_2(l) = c_3 (N-l). \quad (8.10)$$

Consequently, from Eqs. (8.9) and (8.10), the total expected software cost is given by

$$C(l) = C_1(l) + C_2(l). \quad (8.11)$$

From l^* obtained by minimizing l, we can estimate the optimum software release time $\sum_{k=1}^{l^*} \mathrm{E}[X_k]$.

8.5 Reliability Assessment Measures

We can give the following expressions as software reliability assessment measures derived from our hazard rate model:

- *MTBF*
  The MTBF is useful to measure the property of the frequency of software failure-occurrence, and is given by Eq. (8.8).
- *Software reliability*
  Also, the software reliability can be defined as the probability which a software failure does not occur during the time-interval $(t, t+x]$ $(t \geq 0, x \geq 0)$ given that the testing-time of porting-phase is t. The software reliability is given by Eq. (8.7).
- *Porting stability*
  Moreover, we can estimate the porting stability from our hazard rate model. We define the porting stability as the value of model parameter w_1.
- *MSE*
  The MSE (mean square error) can be obtained by dividing the sum of square errors between the observed value, y_k, and its estimated one, $\hat{y}_k$, by the number of data pairs; n. That is,

$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2. \tag{8.12}$$

$\hat{y}_k$ $(k = 1, 2, \ldots, n)$ in Eq. (8.12) is obtained from the estimated value. The mean square error indicates that the selected model fits better to the observed data as MSE becomes small. We compare the proposed flexible hazard rate model for embedded OSS with the following typical conventional hazard rate models:

for Moranda model:

$$z_k(x) = Dc^{k-1} \quad (D > 0,\ 0 < c < 1;\ k = 1,\ 2,\ \ldots), \tag{8.13}$$

$$\mathrm{E}[X_k] = \frac{1}{Dc^{k-1}}. \tag{8.14}$$

for Jelinski–Moranda(J-M) model:

$$z_k(x) = \phi(N - k + 1) \quad (N > 0,\ \phi > 0;\ k = 1,\ 2,\ \ldots,\ N), \tag{8.15}$$

$$\mathrm{E}[X_k] = \frac{1}{\phi(N - k + 1)}. \tag{8.16}$$

The model parameters in Eqs. (8.13) and (8.15) are defined as follows:

D the initial hazard rate for 1st software failure,
c the reduction factor of hazard rate,
ϕ the hazard rate per remaining fault,
N the latent fault in software system.

- *Predicted relative error*
  Furthermore, we adopt the value of the predicted relative error as comparison criteria of goodness-of-fit in our tool. The predicted relative error is defined as a relative error between the predicted and observed values of the software failure-occurrence time-interval. It is given by

$$R_p = \frac{\hat{A}(t_q) - q}{q}, \tag{8.17}$$

  where t_q is the termination time of the porting and q the observed number of faults detected in the time-interval $[0, t_q)$. $\hat{A}(t_q)$ in Eq. (8.17) is the estimated value of the MTBF at the termination time t_q where $A(t)$ is estimated by using the actual data observed up to arbitrary porting time $t_p (0 \leq t_p \leq t_q)$.
- *Optimum software release time*
  Moreover, we can estimate the optimal software release time minimizing the expected total software cost based on our hazard rate model.
- *Total expected software cost*
  We can estimate the total expected software cost in case of the estimated optimum software release time $\sum_{k=1}^{l^*} \mathrm{E}[X_k]$
- *Laplace trend test*
  Also, we use the Laplace trend test [19] of the data set to determine which hazard rate models are useful to investigate. In case of the software failure-occurrence time-interval data, the Laplace trend test statistic $u(i)$ is given by the following equation [19].

$$u(i) = \frac{\frac{1}{i-1}\sum_{n=1}^{i-1}\sum_{j=1}^{n}\theta_j - \frac{\sum_{j=1}^{i}\theta_j}{2}}{\sum_{j=1}^{i}\theta_j\sqrt{\frac{1}{12(i-1)}}}, \tag{8.18}$$

  where i is the software failure number, θ_j means the jth the software failure-occurrence time-interval.

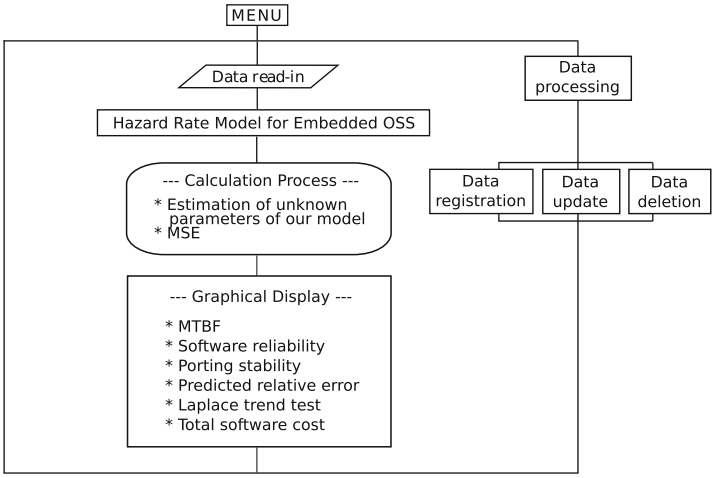

Fig. 8.1 The structure of reliability/portability assessment tool for embedded OSS

8.6 Reliability/Portability Assessment Tool

8.6.1 Specification Requirement

The specification requirement of the reliability/portability assessment tool for embedded OSS are shown as follows:

1. This tool should be operated by clicking the mouse button and typing on the keyboard to input the data through GUI system.
2. An object-oriented language, Java, should be used to implement the program. This tool is developed as a stand-alone application on Windows,[1] Unix,[2] and Macintosh[3] operating system.
3. This tool treats the hazard rate model for embedded OSS, and illustrate the MTBF, software reliability, porting stability, and predicted relative error as software reliability/portability assessment measures.
4. In case of the fault detection time-interval data, the Laplace trend test statistic is illustrated.
5. This tool calculate the MSE for several existing models.

[1] Windows is a registered trademark licensed to Microsoft Corp.

[2] Unix is a registered trademark licensed to the Open group.

[3] Macintosh is a trademark of Macintosh Laboratory, Inc. licensed to Apple Computer, Inc.

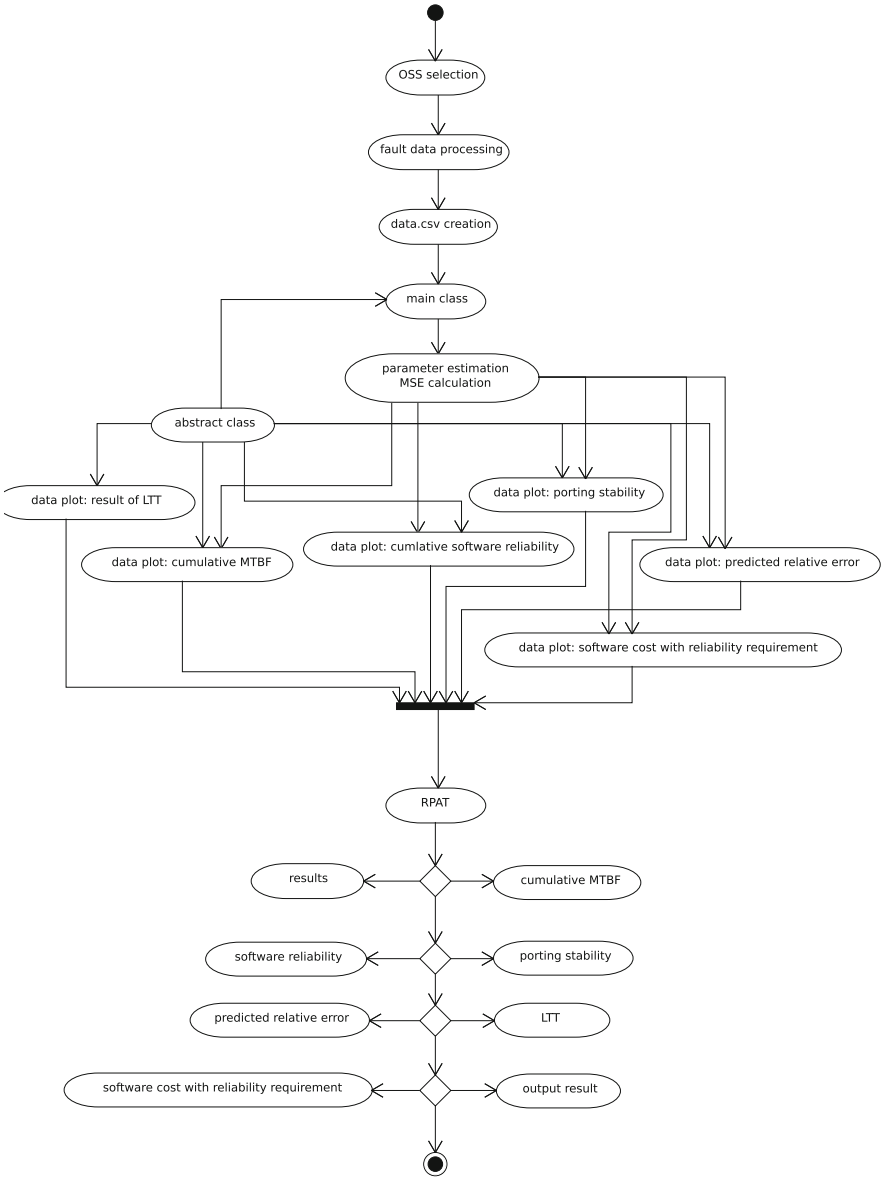

Fig. 8.2 The activity diagram of reliability/portability assessment tool for embedded OSS

8.6.2 Software Reliability Assessment Procedures

The procedures of reliability/portability assessment built into the proposed tool for embedded OSS are shown as follows:

1. This tool processes the data file in terms of the software failure-occurrence time-interval data in the porting-phase of the embedded system for reliability/portability assessment.

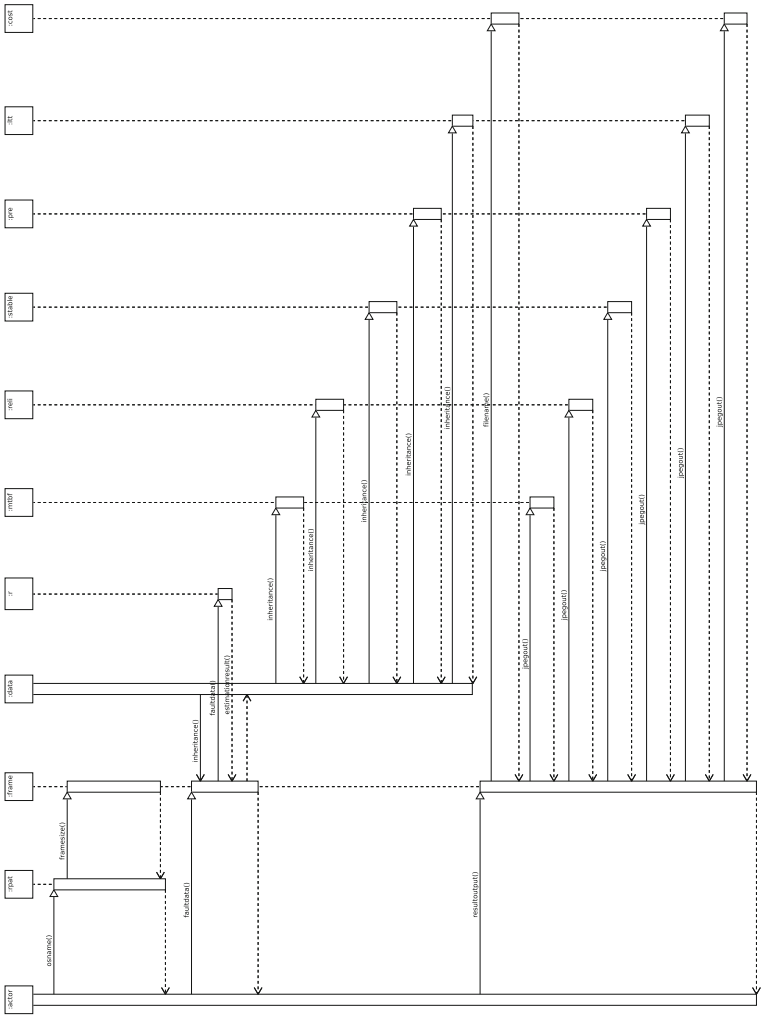

Fig. 8.3 The sequence diagram of reliability/portability assessment tool for embedded OSS

2. Using the data obtained from the porting-phase, we analyze the data for input data.
3. This tool estimates the unknown parameters included in our hazard rate model. We assume as $w_1 = pD$ and $w_2 = (1-p)\phi$ for the simplification technique. Moreover, we define w_1 and w_2 as the scale of the porting stability. Moreover, MSE for several existing model are calculated.
4. This tool illustrates the MTBF, software reliability, porting stability, and predicted relative error as software reliability/portability assessment measures.
5. This tool illustrates the Laplace trend test statistic of the data set.
6. We focus on optimal software release problems based on our hazard rate model for the porting-phase. Especially, the expected total software cost and the optimal software release time minimizing the cost for our model are plotted on the CRT.

This tool is composed of several function components such as fault analysis, estimation of unknown parameters, goodness-of-fit test for the estimated model, graphical representation of fault data, and results of estimation. The structure of reliability/portability assessment tool for embedded OSS is shown in Fig. 8.1. Moreover, we show the activity diagram and the sequence diagram for designing our tool in Figs. 8.2 and 8.3.

References

1. The Apache HTTP Server Project, The Apache Software Foundation, http://httpd.apache.org/
2. Mozilla.org, Mozilla Foundation, http://www.mozilla.org/
3. M.R. Lyu (ed.), *Handbook of Software Reliability Engineering* (IEEE Computer Society Press, Los Alamitos, 1996)
4. J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, 1987)
5. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
6. G.J. Schick, R.W. Wolverton, An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **SE-4**(2), 104–120 (1978)
7. Z. Jelinski, P.B. Moranda, Software reliability research, in *Statistical Computer Performance Evaluation*, ed. by W. Freiberger (Academic Press, New York, 1972), pp. 465–484
8. P.B. Moranda, Event-altered rate models for general reliability analysis. IEEE Trans. Reliab. **R-28**(5), 376–381 (1979)
9. M. Xie, On a generalization of the J-M Model, in *Proceedings of the Reliability '89*, vol. 5, Ba/3/1–5 Ba/3/7 (1989)
10. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach, in *Proceedings of the Workshop on Open Source Software Engineering* (*WOSSE*), vol. 30(4) (2005), pp. 67–72
11. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering* (*FSE-12*) (2004), pp. 263–272
12. J. Norris, Mission-critical development with open source software. IEEE Softw. Mag. **21**(1), 42–49 (2004)
13. Open Handset Alliance, Android, http://www.android.com/
14. E. Andersen, BusyBox, http://www.busybox.net/

15. Y. Tamura, S. Yamada, Comparison of software reliability assessment methods for open source software, in *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems* (*ICPADS2005*), vol. II Fukuoka, Japan, 20–22 July 2005, pp. 488–492
16. Y. Tamura, S. Yamada, A method of user-oriented reliability assessment for open source software and its applications, *Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, 8–11 October 2006, pp. 2185–2190
17. S. Yamada, S. Osaki, Cost-reliability optimal software release policies for software systems. IEEE Trans. Reliab. **R-34**(5), 422–424 (1985)
18. S. Yamada, S. Osaki, Optimal software release policies with simultaneous cost and reliability requirements. Eur. J. Oper. Res. **31**(1), 46–51 (1987)
19. P.A. Keiler, T.A. Mazzuchi, Enhancing the predictive performance of the Goel–Okumoto software reliability growth model, in *Proceedings Annual Reliability and Maintainability Symposium* (IEEE Press, 2000), pp. 106–112

Chapter 9
Reliability Analysis Tool for Open Source Solution

9.1 Introduction

Open source software (OSS) systems which serve as key components of critical infrastructures in the society are still ever-expanding now. Many OSS's are developed in all parts of the world, i.e., Firefox, Apache HTTP server, Linux, Android, etc. Also, there is a growing interest in the next-generation software development paradigm by using network computing technologies such as a cloud computing. The successful experience of adopting the distributed development model in such open source projects includes GNU/Linux operating system, Apache HTTP server, and so on. However, the poor handling of the quality and customer support prohibits the progress of OSS. This chapter focuses on the problems of software quality, which prohibit the progress of OSS. Especially, a large scale open source solution is now attracting attention as the next-generation software development paradigm because of the cost reduction, quick delivery, and work saving. Also, the large scale open source solution has such a unique feature that it is composed of several OSS's. There is no testing phase for the development cycle of each OSS. However, the open source solution has the testing phase such as the binding phase of OSS's with debugging process. Then, it is important to be able to connect several OSS components.

Many software reliability growth models (SRGM's) [1] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective method of dynamic testing management for new distributed development paradigm as typified by the open source project has only a few presented [2–5]. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for open source solution, it is necessary to grasp the situation of registration for bug tracking system, the combination status of OSS's, the degree of maturation of each OSS, etc. Moreover, if the size of the software system is large, the number of faults detected during the testing phase becomes large, and the changes of the number of faults which are detected and removed through each debugging become sufficiently small compared with the initial fault content at the beginning of the testing. There-

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_9

fore, in such a case, a stochastic process model with continuous state space can be used in order to describe the stochastic behavior of the fault-detection process such as the large scale open source solution.

This chapter focuses on an open source solution developed under several OSS's. Also, a useful method of software reliability assessment in open source solution as a typical case of next-generation distributed development paradigm is discussed in this chapter. Then, this chapter propose an SRGM based on stochastic differential equations in order to consider the component collision of OSS. Then, the proposed model assumes that the software fault-detection rate depends on the time, and the software fault-report phenomena on the bug tracking system keep an irregular state. Especially, the reliability analysis tool for open source solution is developed in this chapter. Also, a set of actual software fault-count data is analyzed in order to show numerical examples of software reliability assessment for the open source solution. Moreover, several numerical examples of reliability assessment for each OSS component are shown. Furthermore, several reliability assessment measures are derived from the proposed model. Then, this chapter shows that the developed reliability analysis tool can assist the improvement of quality for an open source solution developed under several OSS's.

9.2 Stochastic Differential Equation Modeling

Let $N(t)$ be the number of faults detected in the open source solution by testing time t ($t \geq 0$). Suppose that $N(t)$ takes on continuous real values. Since latent faults in the open source solution are detected and eliminated during the testing phase, $N(t)$ gradually increases as the testing procedures go on. Thus, under common assumptions for software reliability growth modeling, it is assumed the following linear differential equation:

$$\frac{dN(t)}{dt} = b(t)\{a - N(t)\}, \tag{9.1}$$

where $b(t)$ is the software fault-detection rate at testing time t and a non-negative function.

Generally, it is difficult for software managers to use all functions in open source solution, because the connection state among OSS components is unstable in the testing-phase of open source solution. Considering the characteristic of open source solution, the software fault-report phenomena keep an irregular state in the early stage of testing-phase. Moreover, the addition and deletion of software components are repeated under the development of each OSS system, i.e., the proposed model considers the software failure occurrence phenomenon depending on the time.

Therefore, Eq. (9.1) can be extended to the following stochastic differential equation [6, 7]:

$$\frac{dN(t)}{dt} = \{b(t) + \sigma\gamma(t)\mu(t)\}\{a - N(t)\}, \tag{9.2}$$

where a is the expected number of the initial inherent faults, σ a positive constant representing a magnitude of the irregular fluctuation, $\gamma(t)$ a standardized Gaussian white noise, and $\mu(t)$ the collision level function of OSS component.

Equation (9.2) is extended to the following stochastic differential equation of an Itô type:

$$dN(t) = \left\{b(t) - \frac{1}{2}\sigma^2\mu(t)^2\right\}\{a - N(t)\}dt + \sigma\{a - N(t)\}dW(t), \tag{9.3}$$

where $W(t)$ is a one-dimensional Wiener process which is formally defined as an integration of the white noise $\gamma(t)$ with respect to time t. The Wiener process is a Gaussian process and it has the following properties:

$$\Pr[W(0) = 0] = 1, \tag{9.4}$$

$$\mathrm{E}[W(t)] = 0, \tag{9.5}$$

$$\mathrm{E}[W(t)W(t')] = \mathrm{Min}[t, t'], \tag{9.6}$$

where $\Pr[\cdot]$ and $\mathrm{E}[\cdot]$ represent the probability and expectation, respectively.

By using Itô's formula [6, 7], the solution of Eq. (9.3) under the initial condition $N(0) = 0$ can obtain as follows [8]:

$$N(t) = a\left[1 - \exp\left\{-\int_0^t b(s)ds - \sigma\mu(t)W(t)\right\}\right]. \tag{9.7}$$

Several software reliability measures can be derived by using solution process $N(t)$ in Eq. (9.7).

Moreover, the proposed model is defined as the software fault detection rate per fault in case of $b(t) \equiv b_1(t)$ and $b(t) \equiv b_2(t)$, and the collision level function $\mu(t)$ defined as:

$$b_1(t) = \frac{\frac{dN_e(t)}{dt}}{a_e - N_e(t)} \doteq \frac{\frac{dH_e(t)}{dt}}{a_e - H_e(t)} = b, \tag{9.8}$$

$$b_2(t) = \frac{\frac{dN_s(t)}{dt}}{a_s - N_s(t)} \doteq \frac{\frac{dH_s(t)}{dt}}{a_s - H_s(t)} = \frac{b^2 t}{1 + bt}, \tag{9.9}$$

$$\mu(t) = \exp[-\omega t], \tag{9.10}$$

where $H_e(t)$ and $H_s(t)$ mean the exponential SRGM and the delayed S-shaped SRGM, respectively, based on nonhomogeneous Poisson process (NHPP) [1]. Also, b is the software failure rate per inherent fault, and ω the parameter of stability for the open source solution.

Therefore, the cumulative number of detected faults of these two models with $H_e(t)$ and $H_s(t)$ are obtained as follows, respectively:

$$N_e(t) = a\left[1 - \exp\{-bt - \sigma\mu(t)W(t)\}\right], \tag{9.11}$$

$$N_s(t) = a\left[1 - (1+bt)\exp\{-bt - \sigma\mu(t)W(t)\}\right]. \tag{9.12}$$

9.3 Method of Maximum-Likelihood

In this section, the estimation method of unknown parameters, a, b, ω, and σ in Eq. (9.7) is presented. Let us denote the joint probability distribution function of the process $N(t)$ as

$$\begin{aligned} &P(t_1, y_1; t_2, y_2; \ldots; t_K, y_K) \\ &\quad \equiv \Pr[N(t_1) \le y_1, \ldots, N(t_K) \le y_K | N(t_0) = 0], \end{aligned} \tag{9.13}$$

where $N(t)$ is the cumulative number of faults detected up to the testing time $t\,(t \ge 0)$, and denote its density as

$$\begin{aligned} &p(t_1, y_1; t_2, y_2; \ldots; t_K, y_K) \\ &\quad \equiv \frac{\partial^K P(t_1, y_1; t_2, \ y_2; \ldots; t_K, y_K)}{\partial y_1 \partial y_2 \cdots \partial y_K}. \end{aligned} \tag{9.14}$$

Since $N(t)$ takes on continuous values, the likelihood function l for the observed data $(t_k, y_k)(k = 1, 2, \ldots, K)$ is constructed as follows:

$$l = p(t_1, y_1; t_2, y_2; \ldots; t_K, y_K). \tag{9.15}$$

For convenience in mathematical manipulations, the following logarithmic likelihood function is used:

$$L = \log l. \tag{9.16}$$

The maximum-likelihood estimates, a^*, b^*, ω^*, and σ^* are the values making L in Eq. (9.16) maximize. These can be obtained as the solutions of the following simultaneous likelihood equations [8]:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial \omega} = \frac{\partial L}{\partial \sigma} = 0. \tag{9.17}$$

9.4 Software Reliability Assessment Measures

It is useful for software managers to estimate the expected number of faults detected up to testing time t. The density function of $W(t)$ is given by:

$$f(W(t)) = \frac{1}{\sqrt{2\pi t}} \exp\left\{-\frac{W(t)^2}{2t}\right\}. \tag{9.18}$$

Information on the cumulative number of detected faults in the system is important to estimate the situation of the progress on the software testing procedures. Since $N(t)$ is a random variable in the proposed model, its expected value can be a useful measure. The expected number of faults detected up to time t can be calculated from Eq. (9.7) as follows [8]:

$$\mathrm{E}[N(t)] = a\left[1 - \exp\left\{-\int_0^t b(s)ds + \frac{\sigma^2\mu(t)^2}{2}t\right\}\right]. \tag{9.19}$$

where $\mathrm{E}[N(t)]$ is the expected number of faults detected up to time t. Therefore, the expected number of detected faults, $\mathrm{E}[N_e(t)]$ and $\mathrm{E}[N_s(t)]$ for Eqs. (9.11) and (9.12), are given by the following equations, respectively:

$$\mathrm{E}[N_e(t)] = a\left[1 - \exp\left\{-bt + \frac{\sigma^2\mu(t)^2}{2}t\right\}\right], \tag{9.20}$$

$$\mathrm{E}[N_s(t)] = a\left[1 - (1+bt)\exp\left\{-bt + \frac{\sigma^2\mu(t)^2}{2}t\right\}\right]. \tag{9.21}$$

Also, the expected number of remaining faults at time t can obtain as follows:

$$\mathrm{E}[M(t)] = a - \mathrm{E}[N(t)]. \tag{9.22}$$

Also, the mean time between software failures is useful to measure the property of the frequency of software failure-occurrence. Then, the cumulative MTBF(denoted by $\mathrm{MTBF_C}$) is approximately given by:

$$MTBF_C(t) = \frac{t}{\mathrm{E}[N(t)]}. \tag{9.23}$$

9.5 Comparison of Goodness-of-Fit

The software managers can compare the goodness-of-fit of the proposed stochastic differential equation models (SDE models) in case of $b(t) \equiv b_1(t)$ and $b(t) \equiv b_2(t)$, with the conventional exponential SDE model [8], and the conventional

S-shaped SDE model [8]. The developed tool is adopted the value of the Akaike's information criterion (AIC) as one of comparison criteria of goodness-of-fit. AIC helps software managers to select the optimal model among ones estimated by the method of maximum-likelihood. It is given by

$$\begin{aligned} \mathrm{AIC} = &-2 \cdot (\text{logarithmic maximum-likelihood}) \\ &+2 \cdot (\text{the number of free model parameters}). \end{aligned} \tag{9.24}$$

Differences among their values are significant for AIC, not their values themselves. It can be judged that the model having the smallest AIC fits best to the actual data set when their differences are greater than or equal to 1. However, there are no significant difference among the models in the case where the difference of AIC's are less than 1.

The MSE (mean square error) can be obtained by dividing the sum of square errors between the observed value, y_k, and its estimated one, $\hat{y}_k$, by the number of data pairs; n. That is,

$$\mathrm{MSE} = \frac{1}{n} \sum_{k=1}^{n} (y_k - \hat{y}_k)^2. \tag{9.25}$$

$\hat{y}_k (k = 1, 2, \ldots, n)$ in Eq. (11.10) is obtained from the estimated value. The MSE indicates that the selected model fits better to the observed data as MSE becomes small. The developed tool can compare the proposed SDE models for the open source solution with two typical conventional SDE models [8].

Moreover, the developed tool is adopted the value of the predicted relative error as comparison criteria of goodness-of-fit in the proposed SDE models. The predicted relative error is a relative error of the predicted values and observed data of the number of detected faults that will be discovered by fault-detecting end time t_K when having estimated for any time t_k. If the predicted relative error at time t_k is represented as PRE$[t_k]$, it is calculated by

$$\mathrm{PRE}[t_k] = \frac{\hat{y}(t_k; t_K) - y_K}{y_K}, \tag{9.26}$$

where $\hat{y}(t_k; t_K)$ is the estimated number of detected faults in fault-detecting end time t_K for the observed data until time t_k and y_K observed data of the number of faults by fault-detecting end time t_K.

9.6 Procedures of Reliability Analysis

The procedures of reliability analysis based on the proposed SDE models for an open source solution are shown as follows:

1. The software managers process the fault data in terms of the cumulative number of detected faults in the testing-phase of open source solution for reliability analysis.
2. The software managers estimate the unknown parameters a, b, ω and σ included in the proposed SDE model by using the method of maximum-likelihood.
3. The software managers can compare the proposed SDE models with two typical conventional SDE models [8] by using AIC, MSE, and the predicted relative error.
4. It is useful for the software managers to understand the debugging progress in testing-phase of open source solution development by using the expected number of detected faults, the sample path of detected faults, the expected number of remaining faults, and MTBF's as software reliability assessment measures.

9.7 Reliability Analysis Tool

9.7.1 Specification Requirement

The specification requirement of the reliability analysis tool for open source solution are shown as follows:

1. This tool should be operated by clicking the mouse button and typing on the keyboard to input the data through GUI system.
2. Open source Flex SDK [9] should be used to implement the program. This tool is developed as a stand-alone Adobe Air application on Windows,[1] Unix,[2] and Macintosh OSX[3] operating system. Also, this tool operates as Web application.
3. This tool treats the proposed SDE models for open source solution and the conventional SDE models, and illustrate the expected number of detected faults, the sample path of detected faults, the predicted relative error, the expected number of remaining faults, and the cumulative MTBF as software reliability assessment measures.

9.7.2 Software Reliability Assessment Procedures

The procedures of reliability analysis built into the proposed tool for open source solution are shown as follows:

1. This tool processes the data file in terms of the software fault-detection count data in the testing-phase of the open source solution for reliability analysis.
2. Using the data set obtained from the testing-phase, the data for input data is analyzed.

---

[1] Windows is a registered trademark licensed to Microsoft Corp.

[2] Unix is a registered trademark licensed to the Open group.

[3] Macintosh is a trademark of Macintosh Laboratory, Inc. licensed to Apple Computer, Inc.

3. This tool estimates the unknown parameters included in the proposed SDE models and the conventional SDE models. Also, the estimation results of model parameters are shown on the menu window of the developed tool.
4. This tool illustrates the expected number of detected faults, the sample path of detected faults, the predicted relative error, the expected number of remaining faults, and the cumulative MTBF as software reliability assessment measures.

This tool is composed of several function components such as fault analysis, estimation of unknown parameters, graphical representation of fault data, and results of estimation. Moreover, the activity diagram and the sequence diagram of the developed tool are shown in Figs. 9.1 and 9.2.

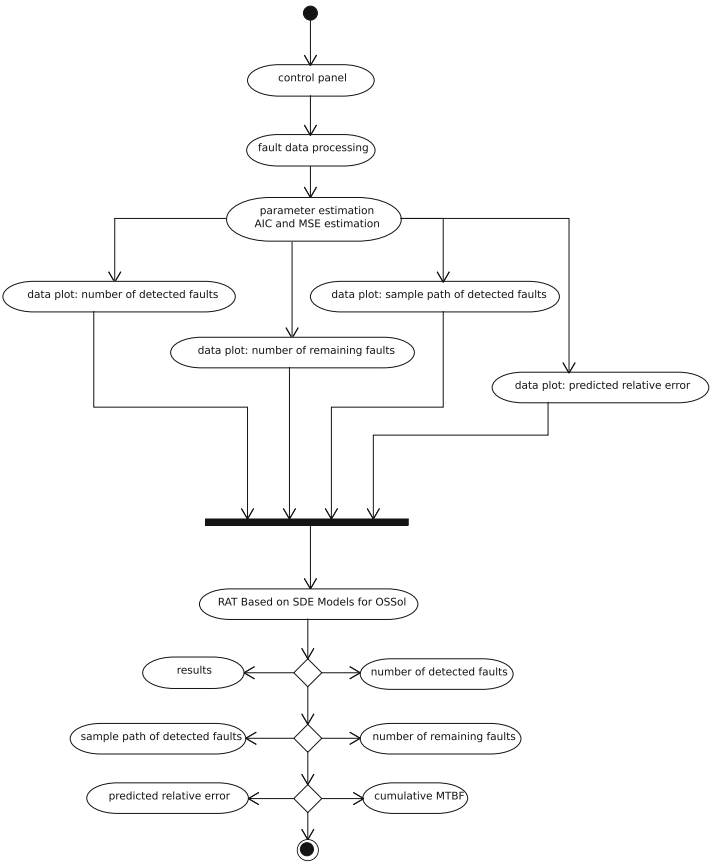

Fig. 9.1 The activity diagram of reliability analysis tool for open source solution

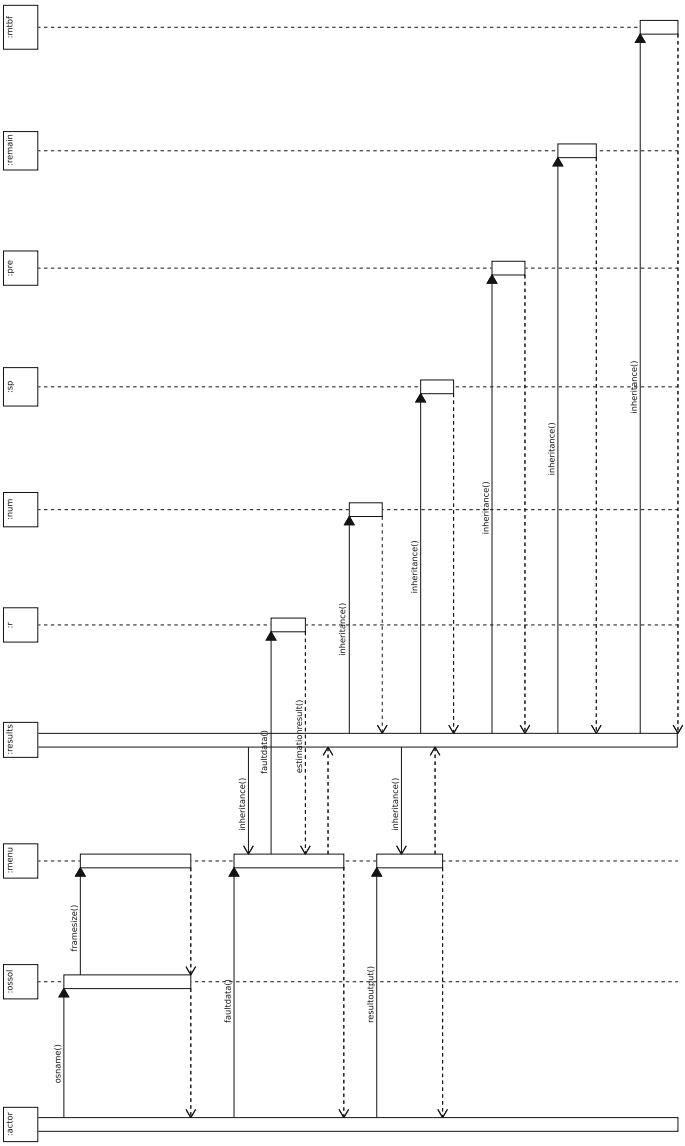

Fig. 9.2 The sequence diagram of reliability analysis tool for open source solution

References

1. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
2. A. MacCormack, J. Rusnak, C.Y. Baldwin, Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Inf. J. Manag. Sci. **52**(7), 1015–1030 (2006)
3. G. Kuk, Strategic interaction and knowledge sharing in the KDE developer mailing list. Inf. J. Manag. Sci. **52**(7), 1031–1042 (2006)
4. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. Proc. Workshop Open Source Softw. Eng. (WOSSE) **30**(4), 67–72 (2005)
5. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems, in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)* (2004), pp. 263–272
6. L. Arnold, *Stochastic Differential Equations-Theory and Applications* (Wiley, New York, 1974)
7. E. Wong, *Stochastic Processes in Information and Systems* (McGraw-Hill, New York, 1971)
8. S. Yamada, M. Kimura, H. Tanaka, S. Osaki, Software reliability measurement and assessment with stochastic differential equations. IEICE Trans. Fundam. **E77-A**(1), 109–116 (1994)
9. Flex.org–Adobe Flex Developer Resource, Adobe Systems Incorporated, http://flex.org/

Chapter 10
Reliability Analysis Tool for Mobile OSS

10.1 Introduction

A cloud computing is now attracting attention as the next-generation software service paradigm because of the cost reduction, quick delivery, and work saving. In particular, third-party software development paradigm has been gaining a lot of attention in various mobile software development area, i.e., Android, OpenStack, and Apache HTTP server, etc. At present, OSS (Open Source Software) systems serve as key components of critical infrastructures in the society. The open source project contains special features so-called software composition by which several geographically-dispersed components are developed in all parts of the world. The successful experience of adopting such open source projects includes Apache HTTP server, MySQL database server, OpenStack cloud software, Firefox Web browser, and GNU/Linux operating system, etc. However, the poor handling of quality problem and customer support has limited the progress of OSS, because the development cycle of OSS has no testing phase.

In particular, a mobile OSS known as one of OSS has been gaining a lot of attention in the embedded system area, i.e., Android [1], BusyBox [2], Firefox OS [3], etc. However, the installer software developed under the third-party developers indirectly effect on the reliability in area of a mobile device. Therefore, it is difficult for many companies to assess the reliability in mobile clouds, because a mobile OSS includes several software versions, the vulnerability issue, the opened source code, the security hole, etc.

Many Software reliability growth models (SRGM's) [4–6] and the related hazard rate models [7–10] have been applied to assess the reliability for quality management and testing progress control of software development. On the other hand, the effective methods assisting dynamic testing management for a new distributed development paradigm as typified by the open source project have only a few presented [11–13]. Also, several research papers [14–18] have been proposed in the area of mobile cloud computing. However, these papers focus on the security, service optimization, secure control, resource allocation technique, etc. The research papers in terms of

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_10

reliability for mobile clouds have only a few presented. In case of considering the effect of the debugging process on entire system in the development of a method of reliability assessment for the software developed under the third-party developers, it is necessary to grasp the situation of installer software, the network traffic, the installed software, etc. Then, it is very important to consider the status of network traffic in terms of the reliability assessment in the following standpoint:

- In case of the open source, the weakness of reliability and security becomes a significant problem via a computer network as shown in Fig. 10.1.
- By using the installer software, the various third-party software are installed via the network.
- In case of the mobile device, the network access devices are frequently used by many software installed via the installer software.

In particular, the bidirectional arrows in Fig. 10.1 show the effects on each factor. For example, the opened source code has an effect on the reliability and security, because the user and cracker can see the source code of software. Also, the change of reliability has an effect on the source code and security, because the source code and security are improved by upgrading to the bug-fix version. Moreover, the security is influenced by the reliability and source code, because the change of the number of detected faults has an effect on the security level, and the opened source code can increase the discoverability of security hole.

Many fault-counting type SRGM's have been applied to assess the reliability for quality management and testing progress control of software development. However, it is difficult to apply the SRGM's for assessing quality/reliability of the software developed under the third-party developers such as OSS. In other words, the testing phase is nonexistent in the open source development paradigm. In fact, there are several SRGM's that can be applied in the above situation, i.e., the Weibull and Log-logistic SRGM's, and so on [4]. In particular, in case the software developed under the third-party developers, it is difficult to assess both the software failure and network

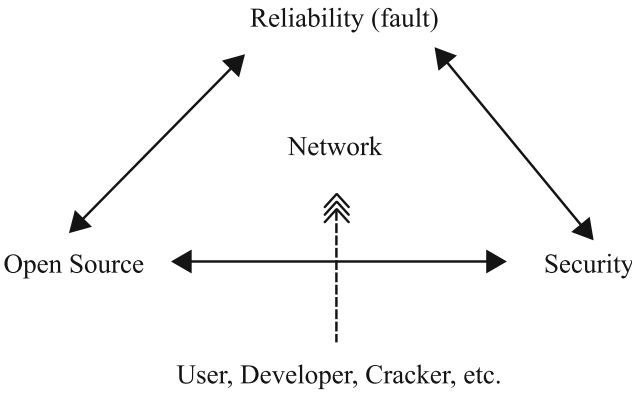

Fig. 10.1 The relationship between reliability and network

traffic affected by using the installer software. As another more challenging aspect of the embedded OSS project, the embedded OSS includes several software components in terms of hardware such as a device driver [19, 20].

From above discussed points, we consider that all factors of *open source software*, *installer software*, and *network access* have an effect on the mobile clouds, directly and indirectly. In other words, the mobile clouds have a deeply complex issue in terms of the reliability. Therefore, it is very important to consider the network environment from the point of view of the reliability for mobile clouds, i.e., it will be able to keep the stable operation of mobile clouds if we can offer several reliability assessment measures considering the network traffic in terms of all factors of *open source software*, *installer software*, and *network access*.

In this chapter, we focus on the method of reliability analysis for the mobile clouds. Then, we propose a method of software reliability analysis based on a hazard rate model and neural network for the mobile clouds. Also, we derive several assessment measures. In particular, we develop an AIR application for reliability analysis based on the proposed method. AIR means Adobe Integrated Runtime. The AIR application can perform on the cross platform, e.g., Windows, Mac OS, and Linux, some mobile operating systems such as BlackBerry Tablet OS, iOS and Android. Our application for reliability assessment is a flexible tool, because our application is developed as AIR application performed on AIR. Moreover, we show the performance examples of the developed AIR application to analyze the method of software reliability assessment for the mobile clouds. Then, we show that the developed AIR application may assist quality improvement for mobile clouds.

10.2 Hazard Rate Model for Mobile OSS

The time-interval between successive software failures of $(k-1)$-th and k-th is represented as the random variable X_k $(k = 1, 2, \ldots)$. Therefore, we can define the hazard rate function $z_k(x)$ at time x during the testing phase for X_k as follows:

$$z_k(x) = w_k(x)\{N - (k-1)\} \qquad (k = 1,\ 2, \ldots,\ N;\ N > 0), \tag{10.1}$$

$$w_k(x) = \phi e^{-p_k} \qquad (\phi > 0,\ -1 < p_k < 1), \tag{10.2}$$

where we can define the each parameter as follows:

$z_k(x)$	the hazard rate for the whole embedded software,
$w_k(x)$	the hazard rate per inherent fault considering the network traffic density,
N	the number of latent faults,
ϕ	the hazard rate per inherent fault,
p_k	the changing rate of the network traffic density.

Equation (10.1) means the hazard rate for a software failure-occurrence phenomenon for the embedded software. Also, we assume that the hazard rate per inherent fault exponentially depends on the network traffic density in terms of the number of

software failures k as shown in Fig. 7.1. In particular, the network traffic changes with the number of installed software and the addition of device drivers. Our model can describe the fault-detection phenomenon considering the network environment by the changing rate of network traffic density p_k included in Eq. (10.2). Thus, the embedded software shows a reliability regression trend if p_k is negative value. On the other hand, the embedded software shows a reliability growth trend if p_k is positive value [21].

10.3 Reliability Assessment Measures

From the hazard rate model for mobile OSS, the software reliability is given by the following equation:

$$R_k(x) = \exp\left[- \left\{ \phi e^{-p_k} \{N - (k-1)\} \right\} \cdot x \right]. \tag{10.3}$$

Moreover, the MTBF is given as follows:

$$\mathrm{E}[X_k] = \frac{1}{\phi e^{-p_k} \{N - (k-1)\}}. \tag{10.4}$$

10.4 Parameter Estimation

We assume that $p_k (k = 1,\ 2, \ldots,\ n)$ are estimated by using neural network. Then, the mean value of the estimated $\widehat{p_k}$ is used in the estimation of parameters N and ϕ. The procedures of reliability analysis based on the proposed model for mobile clouds are shown as follows:

1. The software managers process the data file in terms of the software failure-occurrence time data and the network traffic data in the testing phase of the embedded software for reliability analysis.
2. The software managers estimate the unknown parameters $p_k (k = 1,\ 2, \ldots,\ n)$ included in our hazard rate model by using the neural network.
3. The software managers estimate the unknown parameters N and ϕ included in our hazard rate model by using the method of maximum-likelihood.
4. It is useful for the software managers to understand the debugging progress in testing phase of mobile clouds by using the MTBF in Eq. (5.10) and software reliability in Eq. (5.9) considering the change of network traffic as software reliability assessment measures.

Above described methods of the parameter estimation are shown in the following subsections.

10.5 AIR Application for Reliability Analysis Considering the User Experience Design

10.5.1 Specification Requirement

The specification requirement of the reliability analysis tool for mobile clouds are shown as follows:

1. This tool should be operated by clicking the mouse button and typing on the keyboard to input the data through GUI system. In particular, the user experience design is adopted as the important element of our tool.
2. Open source Apache Flex SDK [22] should be used to implement the program. This tool is developed as a stand-alone Adobe AIR application on Windows,[1] Unix,[2] and Mac OS X[3] operating system. Also, this tool operates as Web application.
3. The method of maximum-likelihood is used as the estimation of unknown parameters in our model. Also, the neural network based on typical time series analysis is used as the estimation of the network traffic density. Then, we assume that the input data $x_i (i = 1, 2, \ldots, t)$ is the set of network traffic data at the number of software failures k. Then, we assume the training phase as follows:

$$\begin{aligned} &\text{Input Data } (x_1,\ x_2, \ldots,\ x_{k-1}),\\ &\text{Training Data } (x_2,\ x_3, \ldots,\ x_k). \end{aligned}$$

   Moreover, we consider the prediction phase based on the learned weight parameters as follows:

$$\begin{aligned} &\text{Input Data } (x_2,\ x_3, \ldots,\ x_k),\\ &\text{Prediction Data } (x_3,\ x_4, \ldots,\ x_{k+1}). \end{aligned}$$

   The network traffic data sets x_{k+2} after the number of software failures $k + 1$ are repeatedly estimated by using the previous data sets as the input data sets.
4. This tool treats the proposed hazard rate model considering the network traffic for mobile clouds, and illustrate the MTBF, the network traffic, and the software reliability as software reliability assessment measures.

[1] Windows is a registered trademark licensed to Microsoft Corp.

[2] Unix is a registered trademark licensed to the Open group.

[3] Macintosh is a trademark of Macintosh Laboratory, Inc. licensed to Apple Computer, Inc.

10.5.2 User Experience Design

It is known the following items as the elements of user experience design.

> "The elements of user experience design"
> Visual Design; Information Architecture; Information; Structuring, Organization and Labeling; Finding and Managing; Interaction Design; Usability; Accessibility; Human-Computer Interaction.

We focus on the "Visual Design" and "Interaction Design" as the user experience design. "Visual Design" and "Interaction Design" can easily implement by using Flex programming language, because Flex includes the following effect components.

- Blur: <mx:Blur — />
- Fade: <mx:Fade — />
- Glow: <mx:Glow — />
- SeriesSlide: <mx:SeriesSlide — />

We develop the dynamic reliability analysis tool based on "Visual Design" and "Interaction Design" by using the animation effects of Flex.

References

1. Open Handset Alliance, Android, http://www.android.com/
2. E. Andersen, BusyBox, http://www.busybox.net/
3. Firefox OS, Marketplace, Android–Partners–mozilla.org, Mozilla Foundation, http://www.mozilla.org/firefoxos/
4. M.R. Lyu (ed.), *Handbook of Software Reliability Engineering* (IEEE Computer Society Press, Los Alamitos, 1996)
5. J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application* (McGraw-Hill, New York, 1987)
6. S. Yamada, *Software Reliability Modeling: Fundamentals and Applications* (Springer, Tokyo, 2014)
7. G.J. Schick, R.W. Wolverton, An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **4**(2), 104–120 (1978)
8. Z. Jelinski, P.B. Moranda, Software reliability research, in *Statistical Computer Performance Evaluation*, (Freiberger, Academic Press, New York, 1972), pp. 465–484
9. P.B. Moranda, Event–altered rate models for general reliability analysis. IEEE Trans. Reliab. **28**(5), 376–381 (1979)
10. M. Xie, On a generalization of the J-M model, in *Proceedings Reliability '89*, 5, Ba/3/1–5 Ba/3/7, 1989
11. Y. Zhoum, J. Davis, Open source software reliability model: an empirical approach. in *Proceedings of the Workshop on Open Source Software Engineering (WOSSE)* vol. 30(4), pp. 67–72 2005
12. P. Li, M. Shaw, J. Herbsleb, B. Ray, P. Santhanam, Empirical evaluation of defect projection models for widely-deployed production software systems. in *Proceeding of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)*, pp. 263–272, 2004
13. J. Norris, Mission-critical development with open source software. IEEE Softw. Mag. **21**(1), 42–49 (2004)

14. J. Park, H.C. Yu, E.Y. Lee, in *Resource allocation techniques based on availability and movement reliability for mobile cloud computing*. Distributed Computing and Internet Technology, Lecture Notes in Computer Science, vol. 7154 (Springer, Berlin, 2012), pp. 263–264
15. H. Suo, Z. Liu, J. Wan, K. Zhou, Security and privacy in mobile cloud computing, in"*Proceedings of the 9th International Wireless Communications and Mobile Computing Conference*, (Cagliari, Italy, 2013), pp. 655–659
16. A. Khalifa, M. Eltoweissy, Collaborative autonomic resource management system for mobile cloud computing, in *Proceedings of the Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, (Valencia, Spain, 2013), pp. 115–121
17. R. Gabner, H.P. Schwefel, K.A. Hummel, G. Haring, Optimal model-based policies for component migration of mobile cloud services, in *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications*, (Cambridge, MA, USA, 2011), pp. 195–202
18. N. Park, in *Secure data access control scheme using type-based re-encryption in cloud environment*, Semantic Methods for Knowledge Management and Communication, Studies in Computational Intelligence, vol. 381 (Springer, Berlin, 2011), pp. 319–327
19. K.Y. Cai, D.B. Hu, C. Bai, H. Hu, T. Jing, Does software reliability growth behavior follow a non-homogeneous poisson process. Inf. Softw. Technol. **50**(12), 1232–1247 (2008)
20. Y. Tamura, S. Yamada, Reliability assessment based on hazard rate model for an embedded OSS porting phase. J. Softw. Test. Verif. Reliab. **23**(1), 77–88 (2013)
21. Y. Tamura, S. Yamada, Reliability analysis based on network traffic for a mobile computing, in *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, (Bangkok, Thailand, 2013), 10–13 December, CD-ROM (RM1)
22. Flex.org–Adobe Flex Developer Resource, Adobe Systems Incorporated, http://flex.org/

Chapter 11
Actual Data and Numerical Examples of OSS Reliability Assessment

11.1 NHPP Model Based on AHP

11.1.1 Reliability Assessment for Each Component

We focus on the Xfce desktop environment [1] which is one of the software system developed under open source project. The Xfce is a lightweight desktop environment for UNIX-like operating systems. It aims to be fast and lightweight, while still being visually appealing and easy to use. The data used in this chapter are collected in the bug tracking system on the website of Xfce which consists of 6 components (called general, other, xfce4, xfdesktop, xffm, and xfwm) in November 2004 [1]. Tables 11.1, 11.2, 11.3, 11.4, 11.5 and 11.6 show actual data sets used in this section. "general" means the panel and utility, etc., "other" is the other components, "xfce4" is core components, "xfdesktop" is the background manager of desktop, "xffm" is the file manager, and "xfwm" is the window manager.

We show the cumulative number of detected faults in each component for actual data in Fig. 11.1. Also, the cumulative number of detected faults in the entire system for the actual data is shown in Fig. 11.2. We use the AIC and the MSE in terms of comparing the goodness-of-fit of the existing SRGM's in Sects. 2.1.2 and 2.13, respectively. The AIC and MSE for each component are shown in Tables 11.7 and 11.8, respectively.

The estimated result of the weight parameter $p_i (i = 1, 2, \ldots, n)$ for each component based on the AHP in Sect. 2.15 is shown in Table 11.9. Especially, the adopted evaluation criteria are the importance level of faults detected for each component (Severity), the fault repairer (Assigned to), and the fault reporter (Reporter). From Table 11.9, we can find that the level of importance for "other" component is the largest. On the other hand, we can find that the level of importance for "general" component is the smallest.

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_11

Table 11.1 The actual data of general component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2003/11/16	2	2003/12/26	18
2003/11/17	3	2003/12/27	19
2003/11/18	5	2003/12/28	19
2003/11/19	7	2003/12/29	20
2003/11/20	7	2003/12/30	20
2003/11/21	8	2003/12/31	20
2003/11/22	8	2004/01/01	20
2003/11/23	9	2004/01/02	20
2003/11/24	9	2004/01/03	20
2003/11/25	9	2004/01/04	20
2003/11/26	9	2004/01/05	20
2003/11/27	9	2004/01/06	20
2003/11/28	9	2004/01/07	20
2003/11/29	9	2004/01/08	21
2003/11/30	9	2004/01/09	21
2003/12/01	9	2004/01/10	22
2003/12/02	9	2004/01/11	22
2003/12/03	9	2004/01/12	22
2003/12/04	9	2004/01/13	23
2003/12/05	9	2004/01/14	24
2003/12/06	11	2004/01/15	25
2003/12/07	11	2004/01/16	25
2003/12/08	12	2004/01/17	25
2003/12/09	13	2004/01/18	25
2003/12/10	13	2004/01/19	26
2003/12/11	13	2004/01/20	26
2003/12/12	14	2004/01/21	26
2003/12/13	14	2004/01/22	27
2003/12/14	14	2004/01/23	27
2003/12/15	14	2004/01/24	27
2003/12/16	14	2004/01/25	30
2003/12/17	16	2004/01/26	30
2003/12/18	16	2004/01/27	31
2003/12/19	17	2004/01/28	31
2003/12/20	17	2004/01/29	31
2003/12/21	17	2004/01/30	31
2003/12/22	17	2004/01/31	34
2003/12/23	17	2004/02/01	34
2003/12/24	17	2004/02/02	34
2003/12/25	17	2004/02/03	34

(continued)

Table 11.1 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/02/04	36	2004/03/13	47
2004/02/05	36	2004/03/14	47
2004/02/06	36	2004/03/15	47
2004/02/07	36	2004/03/16	47
2004/02/08	37	2004/03/17	48
2004/02/09	37	2004/03/18	54
2004/02/10	37	2004/03/19	55
2004/02/11	37	2004/03/20	55
2004/02/12	37	2004/03/21	56
2004/02/13	37	2004/03/22	56
2004/02/14	37	2004/03/23	56
2004/02/15	38	2004/03/24	56
2004/02/16	38	2004/03/25	57
2004/02/17	38	2004/03/26	57
2004/02/18	39	2004/03/27	60
2004/02/19	39	2004/03/28	60
2004/02/20	39	2004/03/29	60
2004/02/21	39	2004/03/30	62
2004/02/22	39	2004/03/31	62
2004/02/23	41	2004/04/01	62
2004/02/24	42	2004/04/02	62
2004/02/25	42	2004/04/03	62
2004/02/26	42	2004/04/04	62
2004/02/27	44	2004/04/05	62
2004/02/28	44	2004/04/06	62
2004/02/29	44	2004/04/07	62
2004/03/01	44	2004/04/08	63
2004/03/02	44	2004/04/09	63
2004/03/03	44	2004/04/10	63
2004/03/04	45	2004/04/11	63
2004/03/05	46	2004/04/12	64
2004/03/06	47	2004/04/13	64
2004/03/07	47	2004/04/14	64
2004/03/08	47	2004/04/15	65
2004/03/09	47	2004/04/16	65
2004/03/10	47	2004/04/17	66
2004/03/11	47	2004/04/18	67
2004/03/12	47	2004/04/19	67

(continued)

Table 11.1 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/04/20	67	2004/05/29	82
2004/04/21	68	2004/05/30	82
2004/04/22	68	2004/05/31	84
2004/04/23	68	2004/06/01	88
2004/04/24	69	2004/06/02	88
2004/04/25	69	2004/06/03	89
2004/04/26	69	2004/06/04	89
2004/04/27	69	2004/06/05	89
2004/04/28	69	2004/06/06	90
2004/04/29	69	2004/06/07	92
2004/04/30	69	2004/06/08	92
2004/05/01	69	2004/06/09	93
2004/05/02	70	2004/06/10	94
2004/05/03	72	2004/06/11	94
2004/05/04	72	2004/06/12	95
2004/05/05	72	2004/06/13	95
2004/05/06	72	2004/06/14	96
2004/05/07	73	2004/06/15	96
2004/05/08	73	2004/06/16	96
2004/05/09	73	2004/06/17	96
2004/05/10	73	2004/06/18	97
2004/05/11	73	2004/06/19	97
2004/05/12	74	2004/06/20	98
2004/05/13	75	2004/06/21	100
2004/05/14	75	2004/06/22	100
2004/05/15	76	2004/06/23	104
2004/05/16	76	2004/06/24	104
2004/05/17	77	2004/06/25	106
2004/05/18	78	2004/06/26	106
2004/05/19	78	2004/06/27	107
2004/05/20	78	2004/06/28	108
2004/05/21	79	2004/06/29	108
2004/05/22	81	2004/06/30	108
2004/05/23	81	2004/07/01	108
2004/05/24	81	2004/07/02	108
2004/05/25	82	2004/07/03	108
2004/05/26	82	2004/07/04	108
2004/05/27	82	2004/07/05	108
2004/05/28	82	2004/07/06	108

(continued)

Table 11.1 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/07/07	108	2004/08/15	124
2004/07/08	108	2004/08/16	124
2004/07/09	108	2004/08/17	124
2004/07/10	108	2004/08/18	124
2004/07/11	110	2004/08/19	124
2004/07/12	110	2004/08/20	124
2004/07/13	111	2004/08/21	125
2004/07/14	112	2004/08/22	128
2004/07/15	113	2004/08/23	128
2004/07/16	113	2004/08/24	130
2004/07/17	113	2004/08/25	130
2004/07/18	113	2004/08/26	132
2004/07/19	113	2004/08/27	134
2004/07/20	113	2004/08/28	135
2004/07/21	115	2004/08/29	135
2004/07/22	115	2004/08/30	135
2004/07/23	116	2004/08/31	135
2004/07/24	116	2004/09/01	135
2004/07/25	116	2004/09/02	135
2004/07/26	118	2004/09/03	135
2004/07/27	118	2004/09/04	136
2004/07/28	119	2004/09/05	136
2004/07/29	119	2004/09/06	136
2004/07/30	119	2004/09/07	136
2004/07/31	120	2004/09/08	136
2004/08/01	120	2004/09/09	137
2004/08/02	121	2004/09/10	138
2004/08/03	122	2004/09/11	138
2004/08/04	122	2004/09/12	139
2004/08/05	123	2004/09/13	140
2004/08/06	123	2004/09/14	140
2004/08/07	124	2004/09/15	140
2004/08/08	124	2004/09/16	141
2004/08/09	124	2004/09/17	142
2004/08/10	124	2004/09/18	142
2004/08/11	124	2004/09/19	142
2004/08/12	124	2004/09/20	142
2004/08/13	124	2004/09/21	142
2004/08/14	124	2004/09/22	142

(continued)

Table 11.1 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/09/23	143	2004/10/22	151
2004/09/24	143	2004/10/23	152
2004/09/25	143	2004/10/24	152
2004/09/26	144	2004/10/25	152
2004/09/27	144	2004/10/26	152
2004/09/28	144	2004/10/27	152
2004/09/29	144	2004/10/28	152
2004/09/30	144	2004/10/29	152
2004/10/01	144	2004/10/30	153
2004/10/02	144	2004/10/31	153
2004/10/03	144	2004/11/01	155
2004/10/04	145	2004/11/02	159
2004/10/05	149	2004/11/03	159
2004/10/06	150	2004/11/04	160
2004/10/07	150	2004/11/05	161
2004/10/08	150	2004/11/06	161
2004/10/09	150	2004/11/07	161
2004/10/10	150	2004/11/08	162
2004/10/11	150	2004/11/09	162
2004/10/12	150	2004/11/10	162
2004/10/13	150	2004/11/11	162
2004/10/14	150	2004/11/12	163
2004/10/15	151	2004/11/13	163
2004/10/16	151	2004/11/14	163
2004/10/17	151	2004/11/15	164
2004/10/18	151	2004/11/16	164
2004/10/19	151	2004/11/17	164
2004/10/20	151	2004/11/18	165
2004/10/21	151		

Table 11.2 The actual data of other component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/09/25	1	2004/10/23	14
2004/09/26	2	2004/10/24	14
2004/09/27	2	2004/10/25	14
2004/09/28	2	2004/10/26	14
2004/09/29	2	2004/10/27	14
2004/09/30	2	2004/10/28	14
2004/10/01	2	2004/10/29	14
2004/10/02	2	2004/10/30	14
2004/10/03	3	2004/10/31	15
2004/10/04	4	2004/11/01	19
2004/10/05	5	2004/11/02	19
2004/10/06	5	2004/11/03	20
2004/10/07	7	2004/11/04	20
2004/10/08	9	2004/11/05	21
2004/10/09	9	2004/11/06	21
2004/10/10	9	2004/11/07	21
2004/10/11	12	2004/11/08	21
2004/10/12	12	2004/11/09	22
2004/10/13	12	2004/11/10	22
2004/10/14	13	2004/11/11	24
2004/10/15	13	2004/11/12	24
2004/10/16	13	2004/11/13	24
2004/10/17	13	2004/11/14	24
2004/10/18	13	2004/11/15	25
2004/10/19	13	2004/11/16	27
2004/10/20	13	2004/11/17	28
2004/10/21	13	2004/11/18	29
2004/10/22	14	2004/11/19	30

Table 11.3 The actual data of xfce4 component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2003/11/12	1	2003/12/21	14
2003/11/13	1	2003/12/22	14
2003/11/14	1	2003/12/23	14
2003/11/15	2	2003/12/24	14
2003/11/16	2	2003/12/25	14
2003/11/17	4	2003/12/26	14
2003/11/18	5	2003/12/27	14
2003/11/19	6	2003/12/28	14
2003/11/20	6	2003/12/29	14
2003/11/21	6	2003/12/30	14
2003/11/22	6	2003/12/31	14
2003/11/23	6	2004/01/01	14
2003/11/24	6	2004/01/02	14
2003/11/25	6	2004/01/03	14
2003/11/26	6	2004/01/04	14
2003/11/27	6	2004/01/05	14
2003/11/28	6	2004/01/06	15
2003/11/29	6	2004/01/07	15
2003/11/30	6	2004/01/08	15
2003/12/01	6	2004/01/09	15
2003/12/02	6	2004/01/10	15
2003/12/03	7	2004/01/11	15
2003/12/04	8	2004/01/12	15
2003/12/05	8	2004/01/13	15
2003/12/06	10	2004/01/14	15
2003/12/07	10	2004/01/15	15
2003/12/08	11	2004/01/16	15
2003/12/09	11	2004/01/17	16
2003/12/10	11	2004/01/18	16
2003/12/11	11	2004/01/19	16
2003/12/12	12	2004/01/20	16
2003/12/13	12	2004/01/21	17
2003/12/14	12	2004/01/22	17
2003/12/15	12	2004/01/23	17
2003/12/16	12	2004/01/24	17
2003/12/17	14	2004/01/25	17
2003/12/18	14	2004/01/26	17
2003/12/19	14	2004/01/27	17
2003/12/20	14	2004/01/28	17

(continued)

Table 11.3 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/01/29	18	2004/03/08	22
2004/01/30	18	2004/03/09	22
2004/01/31	19	2004/03/10	22
2004/02/01	19	2004/03/11	22
2004/02/02	20	2004/03/12	22
2004/02/03	20	2004/03/13	22
2004/02/04	20	2004/03/14	22
2004/02/05	21	2004/03/15	22
2004/02/06	21	2004/03/16	22
2004/02/07	21	2004/03/17	22
2004/02/08	21	2004/03/18	23
2004/02/09	21	2004/03/19	24
2004/02/10	21	2004/03/20	24
2004/02/11	21	2004/03/21	24
2004/02/12	21	2004/03/22	24
2004/02/13	21	2004/03/23	24
2004/02/14	21	2004/03/24	24
2004/02/15	21	2004/03/25	24
2004/02/16	21	2004/03/26	24
2004/02/17	21	2004/03/27	25
2004/02/18	21	2004/03/28	26
2004/02/19	21	2004/03/29	26
2004/02/20	22	2004/03/30	26
2004/02/21	22	2004/03/31	26
2004/02/22	22	2004/04/01	26
2004/02/23	22	2004/04/02	26
2004/02/24	22	2004/04/03	26
2004/02/25	22	2004/04/04	26
2004/02/26	22	2004/04/05	26
2004/02/27	22	2004/04/06	26
2004/02/28	22	2004/04/07	26
2004/02/29	22	2004/04/08	26
2004/03/01	22	2004/04/09	26
2004/03/02	22	2004/04/10	26
2004/03/03	22	2004/04/11	26
2004/03/04	22	2004/04/12	26
2004/03/05	22	2004/04/13	26
2004/03/06	22	2004/04/14	26
2004/03/07	22	2004/04/15	27

(continued)

Table 11.3 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/04/16	27	2004/05/25	35
2004/04/17	27	2004/05/26	35
2004/04/18	27	2004/05/27	36
2004/04/19	27	2004/05/28	36
2004/04/20	27	2004/05/29	36
2004/04/21	27	2004/05/30	36
2004/04/22	27	2004/05/31	36
2004/04/23	27	2004/06/01	38
2004/04/24	27	2004/06/02	38
2004/04/25	28	2004/06/03	38
2004/04/26	28	2004/06/04	38
2004/04/27	28	2004/06/05	38
2004/04/28	28	2004/06/06	38
2004/04/29	28	2004/06/07	38
2004/04/30	28	2004/06/08	38
2004/05/01	28	2004/06/09	38
2004/05/02	28	2004/06/10	38
2004/05/03	29	2004/06/11	38
2004/05/04	29	2004/06/12	38
2004/05/05	29	2004/06/13	38
2004/05/06	29	2004/06/14	38
2004/05/07	30	2004/06/15	38
2004/05/08	31	2004/06/16	38
2004/05/09	31	2004/06/17	38
2004/05/10	31	2004/06/18	39
2004/05/11	31	2004/06/19	39
2004/05/12	32	2004/06/20	39
2004/05/13	32	2004/06/21	39
2004/05/14	32	2004/06/22	40
2004/05/15	32	2004/06/23	40
2004/05/16	32	2004/06/24	40
2004/05/17	32	2004/06/25	41
2004/05/18	32	2004/06/26	41
2004/05/19	32	2004/06/27	41
2004/05/20	32	2004/06/28	41
2004/05/21	32	2004/06/29	42
2004/05/22	32	2004/06/30	42
2004/05/23	33	2004/07/01	42
2004/05/24	33	2004/07/02	42

(continued)

Table 11.3 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/07/03	42	2004/08/11	50
2004/07/04	42	2004/08/12	50
2004/07/05	42	2004/08/13	51
2004/07/06	42	2004/08/14	51
2004/07/07	43	2004/08/15	51
2004/07/08	43	2004/08/16	51
2004/07/09	43	2004/08/17	51
2004/07/10	43	2004/08/18	51
2004/07/11	43	2004/08/19	52
2004/07/12	43	2004/08/20	52
2004/07/13	43	2004/08/21	52
2004/07/14	43	2004/08/22	52
2004/07/15	44	2004/08/23	53
2004/07/16	44	2004/08/24	53
2004/07/17	44	2004/08/25	53
2004/07/18	44	2004/08/26	53
2004/07/19	44	2004/08/27	53
2004/07/20	45	2004/08/28	53
2004/07/21	46	2004/08/29	53
2004/07/22	46	2004/08/30	53
2004/07/23	46	2004/08/31	53
2004/07/24	46	2004/09/01	53
2004/07/25	46	2004/09/02	53
2004/07/26	47	2004/09/03	54
2004/07/27	47	2004/09/04	54
2004/07/28	48	2004/09/05	54
2004/07/29	48	2004/09/06	54
2004/07/30	48	2004/09/07	54
2004/07/31	48	2004/09/08	54
2004/08/01	48	2004/09/09	54
2004/08/02	48	2004/09/10	54
2004/08/03	49	2004/09/11	54
2004/08/04	49	2004/09/12	54
2004/08/05	49	2004/09/13	54
2004/08/06	49	2004/09/14	54
2004/08/07	49	2004/09/15	54
2004/08/08	50	2004/09/16	54
2004/08/09	50	2004/09/17	54
2004/08/10	50	2004/09/18	54

(continued)

Table 11.3 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/09/19	54	2004/10/20	79
2004/09/20	54	2004/10/21	80
2004/09/21	55	2004/10/22	81
2004/09/22	56	2004/10/23	81
2004/09/23	56	2004/10/24	81
2004/09/24	56	2004/10/25	81
2004/09/25	56	2004/10/26	81
2004/09/26	60	2004/10/27	82
2004/09/27	61	2004/10/28	82
2004/09/28	61	2004/10/29	82
2004/09/29	61	2004/10/30	82
2004/09/30	61	2004/10/31	83
2004/10/01	62	2004/11/01	83
2004/10/02	63	2004/11/02	85
2004/10/03	64	2004/11/03	87
2004/10/04	66	2004/11/04	88
2004/10/05	70	2004/11/05	90
2004/10/06	71	2004/11/06	90
2004/10/07	72	2004/11/07	90
2004/10/08	72	2004/11/08	90
2004/10/09	73	2004/11/09	91
2004/10/10	74	2004/11/10	91
2004/10/11	76	2004/11/11	91
2004/10/12	76	2004/11/12	91
2004/10/13	76	2004/11/13	91
2004/10/14	76	2004/11/14	91
2004/10/15	76	2004/11/15	92
2004/10/16	77	2004/11/16	92
2004/10/17	78	2004/11/17	95
2004/10/18	78	2004/11/18	96
2004/10/19	78		

Table 11.4 The actual data of xfdesktop component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/07/21	1	2004/08/29	3
2004/07/22	1	2004/08/30	3
2004/07/23	2	2004/08/31	3
2004/07/24	2	2004/09/01	3
2004/07/25	2	2004/09/02	3
2004/07/26	2	2004/09/03	3
2004/07/27	2	2004/09/04	3
2004/07/28	2	2004/09/05	3
2004/07/29	2	2004/09/06	3
2004/07/30	2	2004/09/07	3
2004/07/31	2	2004/09/08	3
2004/08/01	2	2004/09/09	3
2004/08/02	2	2004/09/10	3
2004/08/03	2	2004/09/11	3
2004/08/04	2	2004/09/12	3
2004/08/05	2	2004/09/13	3
2004/08/06	2	2004/09/14	3
2004/08/07	2	2004/09/15	3
2004/08/08	2	2004/09/16	3
2004/08/09	2	2004/09/17	3
2004/08/10	2	2004/09/18	3
2004/08/11	2	2004/09/19	3
2004/08/12	2	2004/09/20	5
2004/08/13	2	2004/09/21	5
2004/08/14	2	2004/09/22	5
2004/08/15	3	2004/09/23	6
2004/08/16	3	2004/09/24	6
2004/08/17	3	2004/09/25	6
2004/08/18	3	2004/09/26	7
2004/08/19	3	2004/09/27	7
2004/08/20	3	2004/09/28	7
2004/08/21	3	2004/09/29	7
2004/08/22	3	2004/09/30	8
2004/08/23	3	2004/10/01	8
2004/08/24	3	2004/10/02	8
2004/08/25	3	2004/10/03	8
2004/08/26	3	2004/10/04	8
2004/08/27	3	2004/10/05	8
2004/08/28	3	2004/10/06	11

(continued)

Table 11.4 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/10/07	11	2004/10/30	17
2004/10/08	11	2004/10/31	18
2004/10/09	11	2004/11/01	21
2004/10/10	11	2004/11/02	22
2004/10/11	11	2004/11/03	22
2004/10/12	12	2004/11/04	22
2004/10/13	12	2004/11/05	22
2004/10/14	12	2004/11/06	22
2004/10/15	12	2004/11/07	22
2004/10/16	14	2004/11/08	22
2004/10/17	14	2004/11/09	22
2004/10/18	15	2004/11/10	22
2004/10/19	15	2004/11/11	22
2004/10/20	16	2004/11/12	25
2004/10/21	16	2004/11/13	25
2004/10/22	16	2004/11/14	25
2004/10/23	16	2004/11/15	25
2004/10/24	16	2004/11/16	27
2004/10/25	16	2004/11/17	27
2004/10/26	16	2004/11/18	27
2004/10/27	17	2004/11/19	27
2004/10/28	17	2004/11/20	28
2004/10/29	17		

Table 11.5 The actual data of xffm component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2003/11/12	1	2003/12/21	13
2003/11/13	1	2003/12/22	13
2003/11/14	1	2003/12/23	14
2003/11/15	4	2003/12/24	14
2003/11/16	4	2003/12/25	15
2003/11/17	4	2003/12/26	16
2003/11/18	4	2003/12/27	17
2003/11/19	5	2003/12/28	18
2003/11/20	5	2003/12/29	18
2003/11/21	5	2003/12/30	19
2003/11/22	5	2003/12/31	20
2003/11/23	5	2004/01/01	20
2003/11/24	5	2004/01/02	20
2003/11/25	6	2004/01/03	20
2003/11/26	6	2004/01/04	22
2003/11/27	6	2004/01/05	22
2003/11/28	6	2004/01/06	22
2003/11/29	6	2004/01/07	22
2003/11/30	6	2004/01/08	22
2003/12/01	7	2004/01/09	22
2003/12/02	8	2004/01/10	22
2003/12/03	8	2004/01/11	22
2003/12/04	8	2004/01/12	22
2003/12/05	8	2004/01/13	23
2003/12/06	9	2004/01/14	23
2003/12/07	9	2004/01/15	28
2003/12/08	9	2004/01/16	30
2003/12/09	9	2004/01/17	33
2003/12/10	11	2004/01/18	33
2003/12/11	11	2004/01/19	33
2003/12/12	12	2004/01/20	33
2003/12/13	13	2004/01/21	35
2003/12/14	13	2004/01/22	35
2003/12/15	13	2004/01/23	35
2003/12/16	13	2004/01/24	35
2003/12/17	13	2004/01/25	35
2003/12/18	13	2004/01/26	35
2003/12/19	13	2004/01/27	35
2003/12/20	13	2004/01/28	35

(continued)

Table 11.5 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/01/29	35	2004/03/08	42
2004/01/30	35	2004/03/09	42
2004/01/31	35	2004/03/10	42
2004/02/01	36	2004/03/11	43
2004/02/02	37	2004/03/12	43
2004/02/03	39	2004/03/13	43
2004/02/04	39	2004/03/14	44
2004/02/05	39	2004/03/15	44
2004/02/06	40	2004/03/16	44
2004/02/07	40	2004/03/17	44
2004/02/08	40	2004/03/18	45
2004/02/09	40	2004/03/19	45
2004/02/10	40	2004/03/20	45
2004/02/11	40	2004/03/21	45
2004/02/12	40	2004/03/22	45
2004/02/13	40	2004/03/23	45
2004/02/14	40	2004/03/24	45
2004/02/15	40	2004/03/25	45
2004/02/16	40	2004/03/26	45
2004/02/17	40	2004/03/27	45
2004/02/18	40	2004/03/28	46
2004/02/19	40	2004/03/29	46
2004/02/20	41	2004/03/30	46
2004/02/21	42	2004/03/31	46
2004/02/22	42	2004/04/01	46
2004/02/23	42	2004/04/02	46
2004/02/24	42	2004/04/03	46
2004/02/25	42	2004/04/04	47
2004/02/26	42	2004/04/05	47
2004/02/27	42	2004/04/06	47
2004/02/28	42	2004/04/07	47
2004/02/29	42	2004/04/08	47
2004/03/01	42	2004/04/09	47
2004/03/02	42	2004/04/10	47
2004/03/03	42	2004/04/11	47
2004/03/04	42	2004/04/12	47
2004/03/05	42	2004/04/13	47
2004/03/06	42	2004/04/14	47
2004/03/07	42	2004/04/15	47

(continued)

Table 11.5 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/04/16	47	2004/05/23	52
2004/04/17	47	2004/05/24	52
2004/04/18	47	2004/05/25	52
2004/04/19	47	2004/05/26	52
2004/04/20	47	2004/05/27	52
2004/04/21	47	2004/05/28	52
2004/04/22	47	2004/05/29	52
2004/04/23	47	2004/05/30	52
2004/04/24	48	2004/05/31	52
2004/04/25	48	2004/06/01	53
2004/04/26	48	2004/06/02	53
2004/04/27	48	2004/06/03	53
2004/04/28	48	2004/06/04	53
2004/04/29	48	2004/06/05	53
2004/04/30	48	2004/06/06	53
2004/05/01	48	2004/06/07	53
2004/05/02	48	2004/06/08	53
2004/05/03	48	2004/06/09	53
2004/05/04	50	2004/06/10	53
2004/05/05	50	2004/06/11	53
2004/05/06	50	2004/06/12	53
2004/05/07	50	2004/06/13	53
2004/05/08	50	2004/06/14	53
2004/05/09	50	2004/06/15	54
2004/05/10	50	2004/06/16	54
2004/05/11	50	2004/06/17	55
2004/05/12	50	2004/06/18	55
2004/05/13	50	2004/06/19	55
2004/05/14	50	2004/06/20	55
2004/05/15	50	2004/06/21	55
2004/05/16	51	2004/06/22	55
2004/05/17	51	2004/06/23	55
2004/05/18	52	2004/06/24	55
2004/05/19	52	2004/06/25	55
2004/05/20	52	2004/06/26	55
2004/05/21	52	2004/06/27	55
2004/05/22	52	2004/06/28	56

(continued)

Table 11.5 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/06/29	56	2004/08/07	60
2004/06/30	56	2004/08/08	60
2004/07/01	56	2004/08/09	60
2004/07/02	56	2004/08/10	60
2004/07/03	56	2004/08/11	60
2004/07/04	56	2004/08/12	60
2004/07/05	56	2004/08/13	60
2004/07/06	57	2004/08/14	60
2004/07/07	57	2004/08/15	60
2004/07/08	57	2004/08/16	60
2004/07/09	57	2004/08/17	60
2004/07/10	57	2004/08/18	60
2004/07/11	57	2004/08/19	60
2004/07/12	57	2004/08/20	60
2004/07/13	57	2004/08/21	60
2004/07/14	57	2004/08/22	60
2004/07/15	57	2004/08/23	61
2004/07/16	57	2004/08/24	61
2004/07/17	58	2004/08/25	61
2004/07/18	58	2004/08/26	61
2004/07/19	58	2004/08/27	61
2004/07/20	58	2004/08/28	61
2004/07/21	59	2004/08/29	61
2004/07/22	59	2004/08/30	61
2004/07/23	59	2004/08/31	61
2004/07/24	59	2004/09/01	61
2004/07/25	59	2004/09/02	61
2004/07/26	60	2004/09/03	61
2004/07/27	60	2004/09/04	61
2004/07/28	60	2004/09/05	61
2004/07/29	60	2004/09/06	61
2004/07/30	60	2004/09/07	61
2004/07/31	60	2004/09/08	61
2004/08/01	60	2004/09/09	61
2004/08/02	60	2004/09/10	61
2004/08/03	60	2004/09/11	61
2004/08/04	60	2004/09/12	61
2004/08/05	60	2004/09/13	61
2004/08/06	60	2004/09/14	61

(continued)

Table 11.5 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/09/15	61	2004/10/17	68
2004/09/16	61	2004/10/18	68
2004/09/17	61	2004/10/19	68
2004/09/18	61	2004/10/20	68
2004/09/19	61	2004/10/21	68
2004/09/20	62	2004/10/22	68
2004/09/21	62	2004/10/23	68
2004/09/22	62	2004/10/24	68
2004/09/23	62	2004/10/25	69
2004/09/24	62	2004/10/26	71
2004/09/25	62	2004/10/27	71
2004/09/26	64	2004/10/28	71
2004/09/27	65	2004/10/29	71
2004/09/28	65	2004/10/30	71
2004/09/29	65	2004/10/31	71
2004/09/30	65	2004/11/01	72
2004/10/01	65	2004/11/02	72
2004/10/02	65	2004/11/03	72
2004/10/03	66	2004/11/04	72
2004/10/04	66	2004/11/05	72
2004/10/05	66	2004/11/06	72
2004/10/06	66	2004/11/07	72
2004/10/07	66	2004/11/08	72
2004/10/08	66	2004/11/09	72
2004/10/09	66	2004/11/10	74
2004/10/10	66	2004/11/11	74
2004/10/11	68	2004/11/12	74
2004/10/12	68	2004/11/13	74
2004/10/13	68	2004/11/14	74
2004/10/14	68	2004/11/15	74
2004/10/15	68	2004/11/16	75
2004/10/16	68		

Table 11.6 The actual data of xfwm4 component in Xfce

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2003/11/15	1	2003/12/24	7
2003/11/16	1	2003/12/25	7
2003/11/17	1	2003/12/26	7
2003/11/18	2	2003/12/27	7
2003/11/19	2	2003/12/28	7
2003/11/20	2	2003/12/29	7
2003/11/21	2	2003/12/30	7
2003/11/22	2	2003/12/31	7
2003/11/23	2	2004/01/01	7
2003/11/24	3	2004/01/02	7
2003/11/25	3	2004/01/03	7
2003/11/26	3	2004/01/04	7
2003/11/27	3	2004/01/05	7
2003/11/28	3	2004/01/06	10
2003/11/29	3	2004/01/07	11
2003/11/30	3	2004/01/08	11
2003/12/01	3	2004/01/09	12
2003/12/02	3	2004/01/10	12
2003/12/03	3	2004/01/11	12
2003/12/04	3	2004/01/12	13
2003/12/05	3	2004/01/13	13
2003/12/06	3	2004/01/14	13
2003/12/07	3	2004/01/15	14
2003/12/08	4	2004/01/16	14
2003/12/09	5	2004/01/17	14
2003/12/10	5	2004/01/18	14
2003/12/11	5	2004/01/19	14
2003/12/12	6	2004/01/20	14
2003/12/13	6	2004/01/21	15
2003/12/14	6	2004/01/22	15
2003/12/15	6	2004/01/23	15
2003/12/16	6	2004/01/24	15
2003/12/17	6	2004/01/25	15
2003/12/18	6	2004/01/26	15
2003/12/19	6	2004/01/27	15
2003/12/20	6	2004/01/28	15
2003/12/21	6	2004/01/29	15
2003/12/22	6	2004/01/30	15
2003/12/23	6	2004/01/31	15

(continued)

Table 11.6 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/02/01	15	2004/03/11	21
2004/02/02	15	2004/03/12	21
2004/02/03	15	2004/03/13	21
2004/02/04	15	2004/03/14	21
2004/02/05	15	2004/03/15	21
2004/02/06	15	2004/03/16	22
2004/02/07	15	2004/03/17	23
2004/02/08	16	2004/03/18	25
2004/02/09	16	2004/03/19	25
2004/02/10	16	2004/03/20	25
2004/02/11	16	2004/03/21	25
2004/02/12	16	2004/03/22	25
2004/02/13	17	2004/03/23	25
2004/02/14	18	2004/03/24	26
2004/02/15	18	2004/03/25	26
2004/02/16	18	2004/03/26	27
2004/02/17	18	2004/03/27	29
2004/02/18	18	2004/03/28	29
2004/02/19	18	2004/03/29	30
2004/02/20	18	2004/03/30	30
2004/02/21	18	2004/03/31	30
2004/02/22	18	2004/04/01	30
2004/02/23	18	2004/04/02	30
2004/02/24	18	2004/04/03	31
2004/02/25	18	2004/04/04	31
2004/02/26	18	2004/04/05	32
2004/02/27	18	2004/04/06	32
2004/02/28	18	2004/04/07	32
2004/02/29	19	2004/04/08	32
2004/03/01	19	2004/04/09	32
2004/03/02	19	2004/04/10	32
2004/03/03	19	2004/04/11	33
2004/03/04	20	2004/04/12	34
2004/03/05	21	2004/04/13	34
2004/03/06	21	2004/04/14	34
2004/03/07	21	2004/04/15	34
2004/03/08	21	2004/04/16	34
2004/03/09	21	2004/04/17	34
2004/03/10	21	2004/04/18	34

(continued)

Table 11.6 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/04/19	34	2004/05/27	36
2004/04/20	34	2004/05/28	36
2004/04/21	34	2004/05/29	36
2004/04/22	34	2004/05/30	36
2004/04/23	34	2004/05/31	36
2004/04/24	34	2004/06/01	36
2004/04/25	34	2004/06/02	36
2004/04/26	34	2004/06/03	36
2004/04/27	34	2004/06/04	36
2004/04/28	34	2004/06/05	37
2004/04/29	34	2004/06/06	37
2004/04/30	34	2004/06/07	37
2004/05/01	34	2004/06/08	37
2004/05/02	34	2004/06/09	37
2004/05/03	35	2004/06/10	37
2004/05/04	35	2004/06/11	37
2004/05/05	35	2004/06/12	38
2004/05/06	35	2004/06/13	38
2004/05/07	35	2004/06/14	38
2004/05/08	35	2004/06/15	38
2004/05/09	35	2004/06/16	39
2004/05/10	35	2004/06/17	39
2004/05/11	35	2004/06/18	39
2004/05/12	35	2004/06/19	39
2004/05/13	35	2004/06/20	39
2004/05/14	35	2004/06/21	39
2004/05/15	36	2004/06/22	39
2004/05/16	36	2004/06/23	39
2004/05/17	36	2004/06/24	39
2004/05/18	36	2004/06/25	39
2004/05/19	36	2004/06/26	39
2004/05/20	36	2004/06/27	40
2004/05/21	36	2004/06/28	41
2004/05/22	36	2004/06/29	41
2004/05/23	36	2004/06/30	41
2004/05/24	36	2004/07/01	41
2004/05/25	36	2004/07/02	41
2004/05/26	36	2004/07/03	41

(continued)

Table 11.6 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/07/04	41	2004/08/11	54
2004/07/05	41	2004/08/12	54
2004/07/06	41	2004/08/13	56
2004/07/07	42	2004/08/14	57
2004/07/08	42	2004/08/15	57
2004/07/09	44	2004/08/16	57
2004/07/10	45	2004/08/17	57
2004/07/11	45	2004/08/18	57
2004/07/12	45	2004/08/19	57
2004/07/13	45	2004/08/20	57
2004/07/14	45	2004/08/21	57
2004/07/15	45	2004/08/22	57
2004/07/16	45	2004/08/23	57
2004/07/17	45	2004/08/24	57
2004/07/18	45	2004/08/25	57
2004/07/19	45	2004/08/26	57
2004/07/20	46	2004/08/27	57
2004/07/21	46	2004/08/28	57
2004/07/22	47	2004/08/29	57
2004/07/23	48	2004/08/30	57
2004/07/24	48	2004/08/31	57
2004/07/25	49	2004/09/01	57
2004/07/26	49	2004/09/02	57
2004/07/27	50	2004/09/03	57
2004/07/28	50	2004/09/04	57
2004/07/29	51	2004/09/05	57
2004/07/30	51	2004/09/06	57
2004/07/31	52	2004/09/07	57
2004/08/01	52	2004/09/08	57
2004/08/02	52	2004/09/09	58
2004/08/03	52	2004/09/10	58
2004/08/04	52	2004/09/11	58
2004/08/05	52	2004/09/12	58
2004/08/06	52	2004/09/13	58
2004/08/07	53	2004/09/14	58
2004/08/08	53	2004/09/15	58
2004/08/09	53	2004/09/16	58
2004/08/10	53	2004/09/17	58

(continued)

Table 11.6 (continued)

Date (Year/Month/Day)	Cumulative number of detected faults	Date (Year/Month/Day)	Cumulative number of detected faults
2004/09/18	58	2004/10/19	76
2004/09/19	58	2004/10/20	76
2004/09/20	58	2004/10/21	76
2004/09/21	59	2004/10/22	79
2004/09/22	59	2004/10/23	79
2004/09/23	59	2004/10/24	79
2004/09/24	59	2004/10/25	79
2004/09/25	60	2004/10/26	79
2004/09/26	62	2004/10/27	81
2004/09/27	63	2004/10/28	81
2004/09/28	63	2004/10/29	82
2004/09/29	63	2004/10/30	82
2004/09/30	63	2004/10/31	83
2004/10/01	64	2004/11/01	88
2004/10/02	64	2004/11/02	89
2004/10/03	64	2004/11/03	89
2004/10/04	66	2004/11/04	90
2004/10/05	66	2004/11/05	90
2004/10/06	67	2004/11/06	91
2004/10/07	68	2004/11/07	91
2004/10/08	70	2004/11/08	92
2004/10/09	71	2004/11/09	93
2004/10/10	71	2004/11/10	93
2004/10/11	71	2004/11/11	93
2004/10/12	73	2004/11/12	94
2004/10/13	75	2004/11/13	94
2004/10/14	75	2004/11/14	94
2004/10/15	75	2004/11/15	95
2004/10/16	76	2004/11/16	96
2004/10/17	76	2004/11/17	96
2004/10/18	76	2004/11/18	97

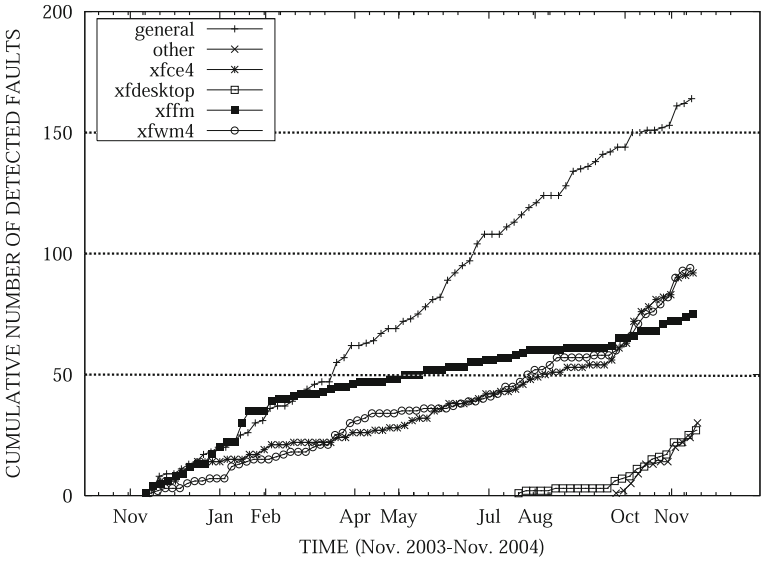

Fig. 11.1 The cumulative number of detected faults in each component for actual data

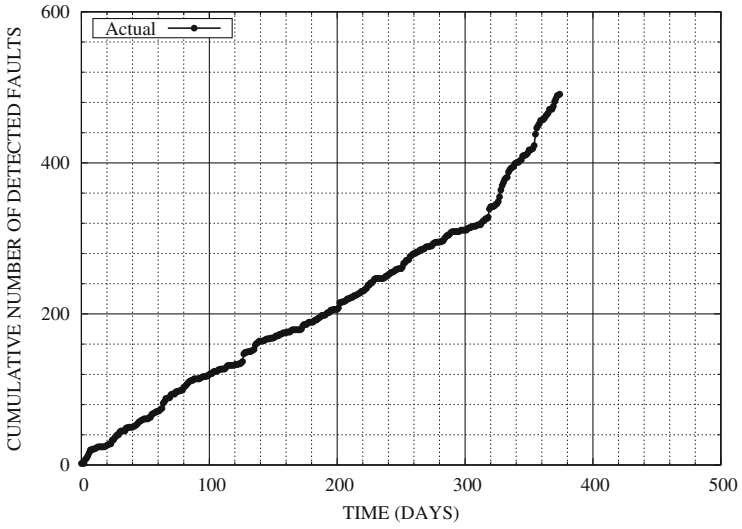

Fig. 11.2 The cumulative number of detected faults in the Xfce for actual data

Table 11.7 Comparison of the AIC for each component

	Exponential SRGM	Inflection S-shaped SRGM
General	675.03	675.80*
Other	112.29*	113.60
xfce4	480.96	466.39*
xfdesktop	151.01	138.69*
xffm	403.51*	1258.4
xfwm	482.64	468.11*

(*means the selected model)

Table 11.8 Comparison of the MSE for each component

	Exponential SRGM	Inflection S-shaped SRGM
General	13.796	8.3644*
Other	3.5658*	3.5482
xfce4	178.13	36.933*
xfdesktop	28.886	1.6298*
xffm	20.400*	858.17
xfwm	149.91	27.499*

(*means the selected model)

Table 11.9 The estimated results of the weight parameter for each component based on AHP

Component	Weight parameter p_i
General	0.055219
Other	0.44820
xfce4	0.091727
xfdesktop	0.18165
xffm	0.11970
xfwm	0.10351

11.1.2 Reliability Assessment for Entire System

On the presupposition that unknown parameters of SRGM's applied to each component are estimated by using the method of maximum-likelihood, we show numerical examples for reliability assessment of Xfce desktop environment. The estimated number of detected faults in Eq. (2.8), $\widehat{\mu}(t)$ is shown in Fig. 11.3. Figure 11.4 shows the estimated software reliability in Eq. (2.10), $\widehat{R}(x|t)$. Moreover, the estimated $\mathrm{MTBF_I}$ in Eq. (2.11), $\widehat{MTBF}_I(t)$ and the estimated $\mathrm{MTBF_C}$ in Eq. (2.12), $\widehat{MTBF}_C(t)$ are also plotted in Figs. 11.5 and 11.6, respectively.

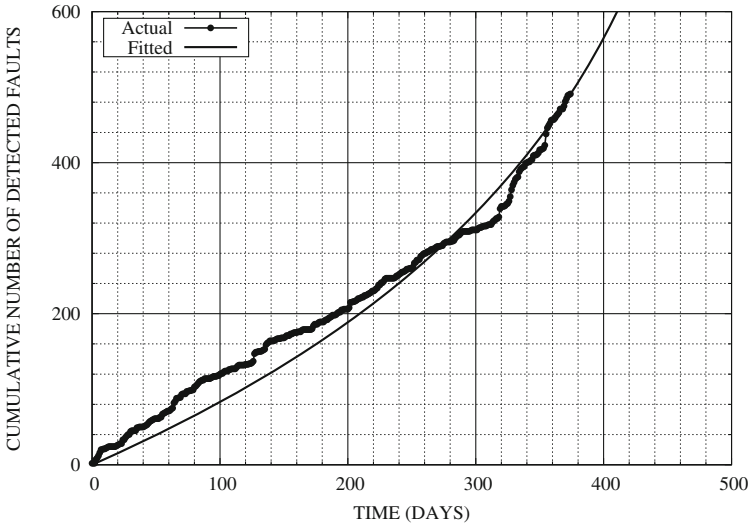

Fig. 11.3 The estimated number of detected faults, $\widehat{\mu}(t)$

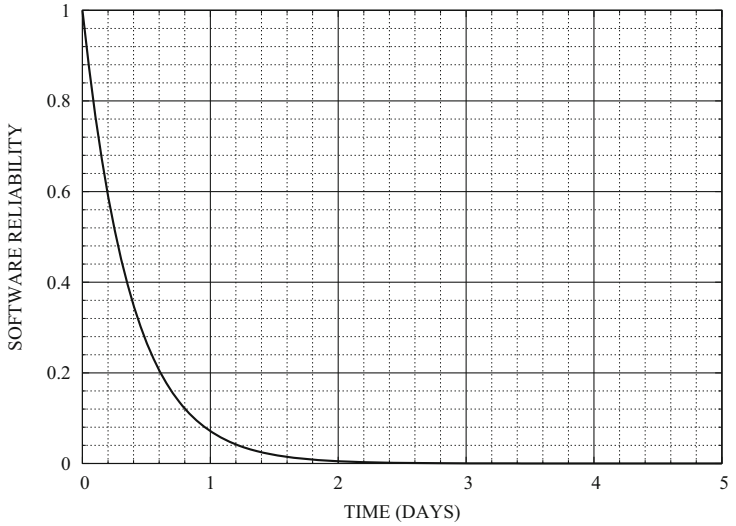

Fig. 11.4 The estimated software reliability, $\widehat{R}(x|t)$

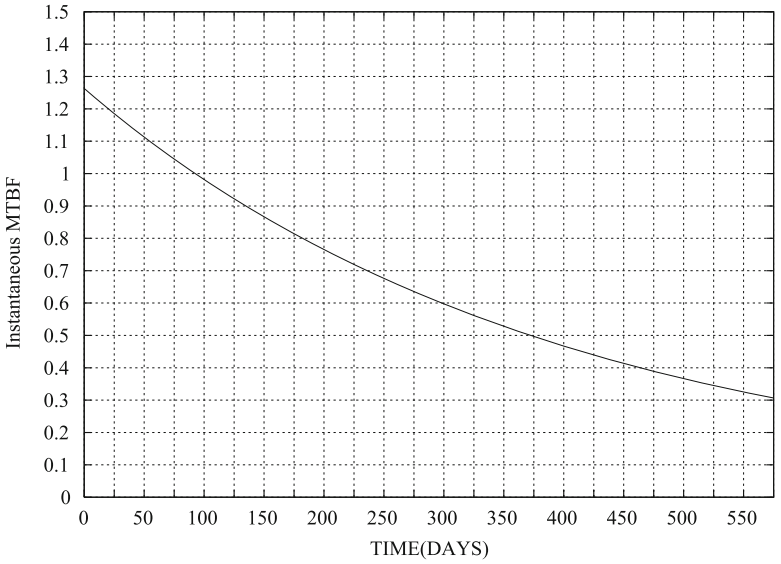

Fig. 11.5 The estimated $\mathrm{MTBF_I}$, $\widehat{MTBF_I}(t)$

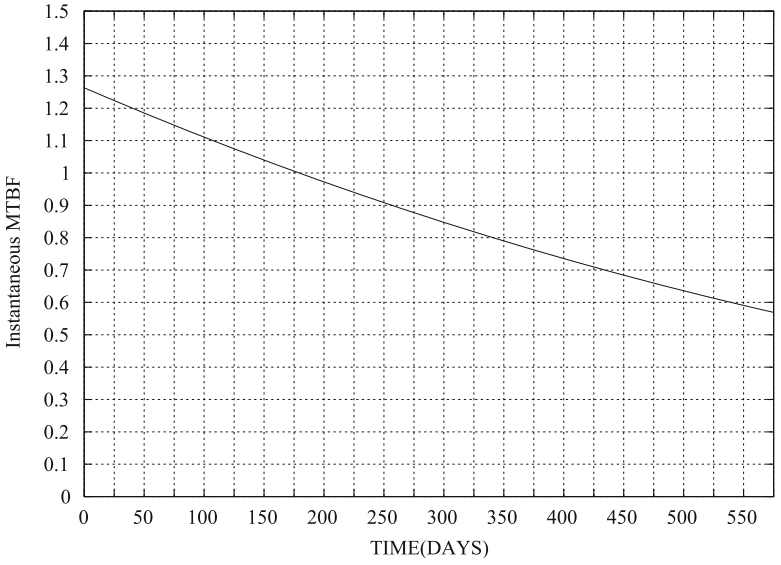

Fig. 11.6 The estimated $\mathrm{MTBF_C}$, $\widehat{MTBF_C}(t)$

11.1.3 Discussion for the Method of Reliability Assessment Based on AHP

We have focused on the Xfce desktop environment which is one of the desktop environment, and discussed the method of reliability assessment for distributed development paradigm based on an open source project. Especially, we have applied SRGM's on AHP which is known as one of the method of decision-making in order to consider the effect of each software component on the reliability of entire system under such distributed development environment. By using the AHP, we have proposed the method of reliability assessment incorporating the interaction among software components. The AHP, exponential SRGM and inflection S-shaped SRGM applied in this section have the simple structure. Therefore, we can easily apply our method to actual distributed software development project.

In case of considering the effect of debugging process on entire system in the development of software reliability assessment methods for distributed development environment, it is necessary to grasp the deeply-intertwined factors. In this chapter, we have shown that our method can grasp such deeply-intertwined factors by using the weight parameters of evaluation criteria for each component in AHP. Additionally, we have presented several numerical examples for the actual data collected in the bug tracking system on the website of Xfce. Moreover, we have given several estimated reliability assessment measures based on the proposed method.

Finally, we have focused on an OSS developed under open source projects. Distributed development environment typified by such open source project will evolve at a rapid pace in the future. Our method is useful as the method of reliability assessment incorporating the importance of each component for entire system.

11.2 NHPP Model Based on ANP

11.2.1 Reliability Assessment for Each Component

We show the cumulative number of detected faults in each component for actual data in Fig. 11.7. We use the Akaike's information criterion (AIC) and the mean square error (MSE) in terms of comparing the goodness-of-fit of existing SRGM in Sect. 3.1.4. The AIC and MSE for each component are shown in Tables 11.10 and 11.11.

The estimated result of weight parameter $p_i (i = 1, 2, \ldots, n)$ for each component based on ANP in Fig. 11.7 is shown in Table 11.12. Especially, the applied evaluation criteria are the importance level of faults detected for each component (Severity), the fault repairer (Assigned to), and the fault reporter(Reporter). From Table 11.12, we find that the level of importance for "other" component is largest. On the other hand, we find that the level of importance for "general" component is the smallest.

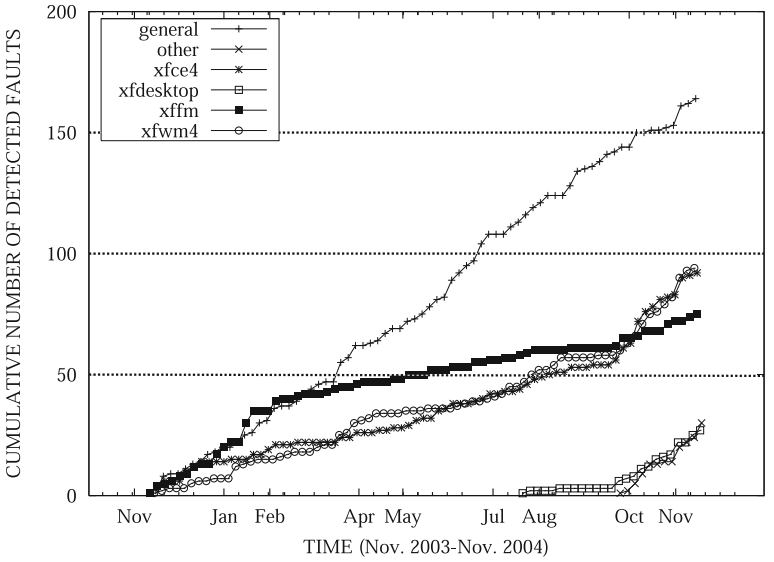

Fig. 11.7 The cumulative number of detected faults in each component for actual data

Table 11.10 Comparison of the AIC for each component

	Exponential SRGM	Inflection S-shaped SRGM
General	675.03	675.80*
Other	112.29*	113.60
xfce4	480.96	466.39*
xfdesktop	151.01	138.69*
xffm	403.51*	1258.4
xfwm	482.64	468.11*

(*means the selected model)

Table 11.11 Comparison of the MSE for each component

	Exponential SRGM	Inflection S-shaped SRGM
General	13.796	8.3644*
Other	3.5658*	3.5482
xfce4	178.13	36.933*
xfdesktop	28.886	1.6298*
xffm	20.400*	858.17
xfwm	149.91	27.499*

(*means the selected model)

Table 11.12 The estimated results of the weight parameter for each component based on ANP

Component	Weight parameter p_i
General	0.0550295
Other	0.444284
xfce4	0.093274
xfdesktop	0.1827720
xffm	0.122944
xfwm	0.101697

11.2.2 Reliability Assessment for Entire System

On the presupposition that the unknown parameters of SRGM applied to each component are estimated by using a maximum-likelihood estimate method, we show numerical examples for reliability assessment of Xfce desktop environment. The estimated cumulative number of detected faults in Eq. (3.9), $\hat{S}(t)$, is shown in Fig. 11.8. Figure 11.9 shows the estimated software reliability in Eq. (3.12), $\hat{R}_c(x|t)$.

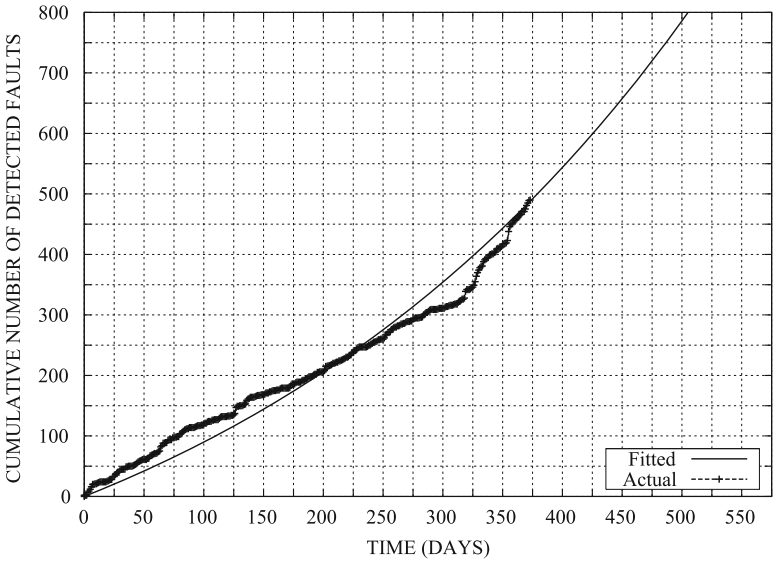

Fig. 11.8 The estimated cumulative number of detected faults, $\hat{S}(t)$

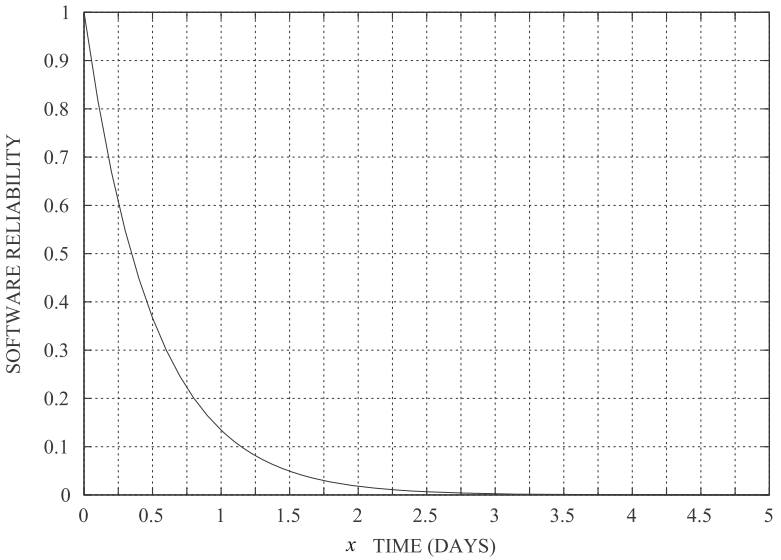

Fig. 11.9 The estimated software reliability, $\hat{R}_e(x|t)$

11.2.3 Discussion for the Method of Reliability Assessment Based on ANP

We have focused on the Xfce desktop environment which is one of the desktop environment, and discussed the method of reliability assessment for distributed development paradigm based on open source project. Especially we have applied ANP which is known as one of the method of decision-making in order to consider the effect of each software component on the reliability of entire system under such distributed development environment. By using ANP, we have proposed the method of reliability assessment incorporating the interaction among each software component. ANP and the inflection S-shaped SRGM applied in this chapter have the simple structure. Therefore, we can easily apply our method to actual distributed software development project.

In case of considering the effect of debugging process on entire system in the development of a method of software reliability assessment for distributed development environment, it is necessary to grasp the deeply-intertwined factors. In this section, we have shown that our method can grasp such deeply-intertwined factors by using weight parameter of evaluation criteria for each component in ANP. Further, we have presented several numerical examples for the actual data collected in the bug tracking system of the website of Xfce. Moreover, we have given several estimated reliability assessment measures based on the proposed method.

11.3 Stochastic Differential Equation Models

11.3.1 Data for Numerical Illustrations

We focus on the Fedora Core Linux [2] which is one of the operating system developed under an open source project. The Fedora project is made up of many small-size projects. Fedora is a set of projects, sponsored by Red Hat and guided by the Fedora Project Board.[1] These projects are developed by a large community of people who strive to provide and maintain the very best in free, open source software and standards.

The fault-count data used in this section are collected in the bug tracking system on the website of Fedora project in October 2006. Especially, we focus on the Kernel component of the Fedora Core Linux. Table 11.13 shows actual data sets used in this section.

11.3.2 Reliability Assessment

The estimated expected number of detected faults in Eq. (4.16), $\widehat{\mathrm{E}}[S(t)]$, is shown in Fig. 11.10. Also, the sample path of the estimated numbers of detected faults in Eq. (4.7), $\widehat{S}(t)$, is shown in Fig. 11.11 approximately.

Figure 11.12 shows the estimated variance of the number of faults in Eq. (4.17), $\widehat{\mathrm{Var}}[S(t)]$. In Fig. 11.12, it is shown that the variance of the number of detected faults grows as the time elapses after the evaluated version of Fedora Core 6 has been released.

Moreover, the estimated $\mathrm{MTBF_I}$ in Eq. (4.19), $\widehat{MTBF}_I(t)$, and the estimated $\mathrm{MTBF_C}$ in Eq. (4.21), $\widehat{MTBF}_C(t)$, are also plotted in Figs. 11.13 and 11.14, respectively. These figures show that the MTBF increases as the operational procedures go on. Figure 11.15 shows the estimated coefficient of variation. Figure 11.15 means a decrease in stability.

11.3.3 Sensitivity Analysis in Terms of Model Parameters

From the results of the former sections, we have verified that our model can be applied to assess quantitatively software reliability in the operational phase of the OSS. In this section, we show some behavior of software reliability assessment measures if we change the parameters σ and α which are the magnitude of the irregular fluctuation and the acceleration parameter of the intensity of initial inherent failure.

[1] Fedora is a trademark of Red Hat, Inc. The Fedora Project is not a supported product of Red Hat, Inc.

Table 11.13 The actual data in Fedora Core Linux

Unit Time (Days)	Cumulative number of detected faults	Unit Time (Days)	Cumulative number of detected faults
0	41	29	90
1	41	30	91
2	41	31	95
3	47	32	97
4	51	33	100
5	56	34	101
6	59	35	102
7	61	36	102
8	62	37	103
9	64	38	104
10	69	39	104
11	71	40	106
12	71	41	109
13	72	42	114
14	72	43	116
15	72	44	119
16	74	45	121
17	77	46	126
18	77	47	127
19	79	48	129
20	81	49	129
21	81	50	129
22	83	51	130
23	86	52	132
24	87	53	133
25	88	54	134
26	90	55	135
27	90	56	136
28	90		

In addition to the case of $\alpha = 0.046482$ and $\sigma = 0.10192$ in the former section, we represent the estimated mean value function with changing the value of parameters σ and α at regular intervals are illustrated in Figs. 11.16 and 11.17, respectively.

Moreover, we show the estimated reliability assessment measures with changing the value of the parameter σ and α at regular intervals are illustrated in Figs. 11.18, 11.19, 11.20 and 11.21, respectively.

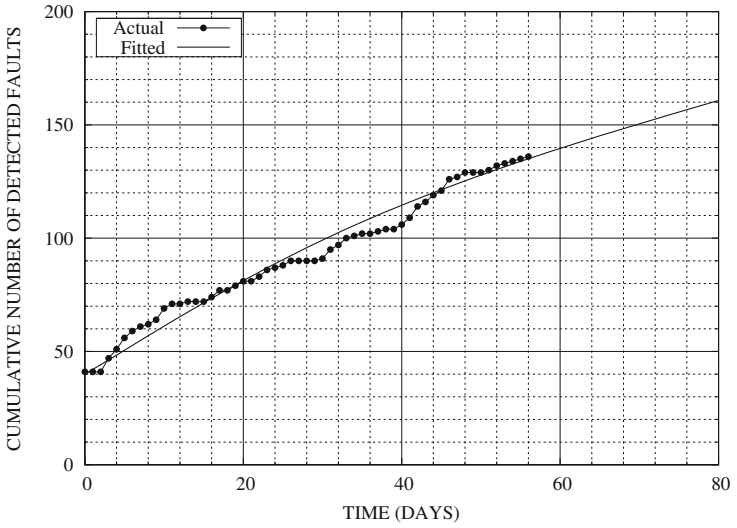

Fig. 11.10 The estimated number of detected faults, $\hat{\mathrm{E}}[S(t)]$

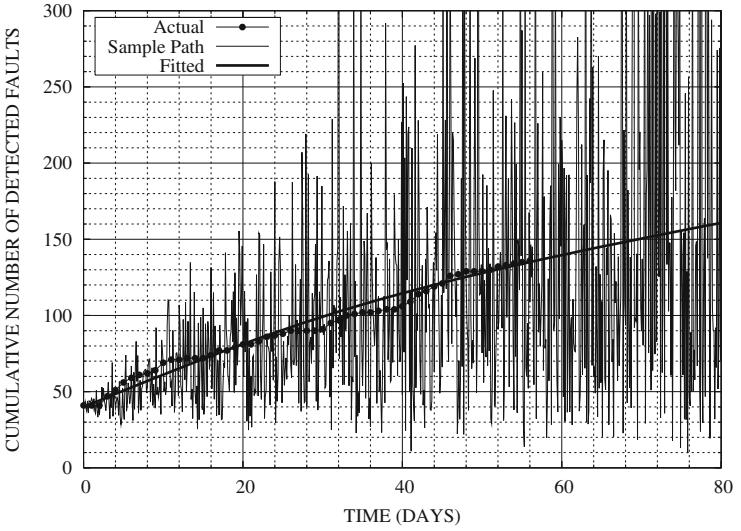

Fig. 11.11 The sample path of the estimated cumulative number of remaining faults, $\widehat{S}(t)$

11.3.4 Results of Goodness-of-Fit Comparison

We show the reliability assessment results for the other OSS in terms of the performance evaluation of our model. We focus on the Apache HTTP server [3] which is the most popular HTTP server developed under an open source project. The Apache

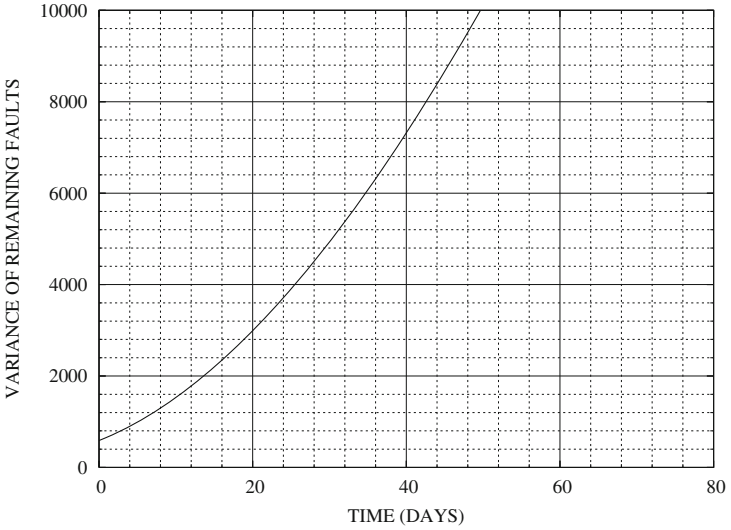

Fig. 11.12 The estimated variance of the number of detected faults, $\widehat{\mathrm{Var}}[S(t)]$

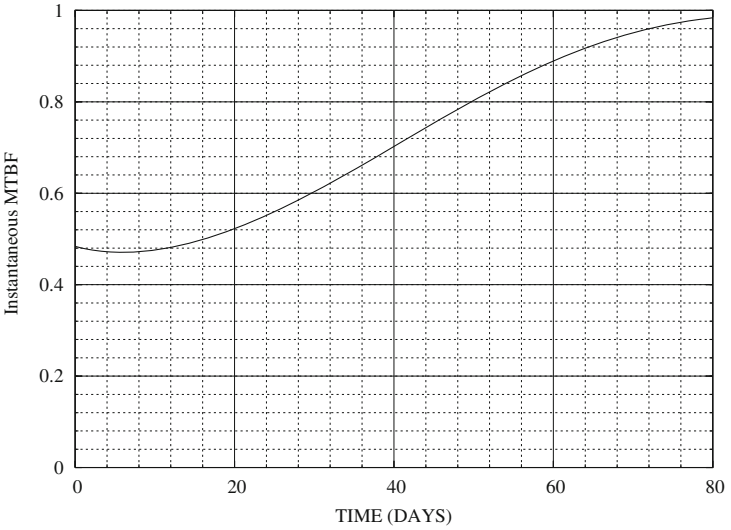

Fig. 11.13 The estimated $\mathrm{MTBF_I}$, $\widehat{MTBF_I}(t)$

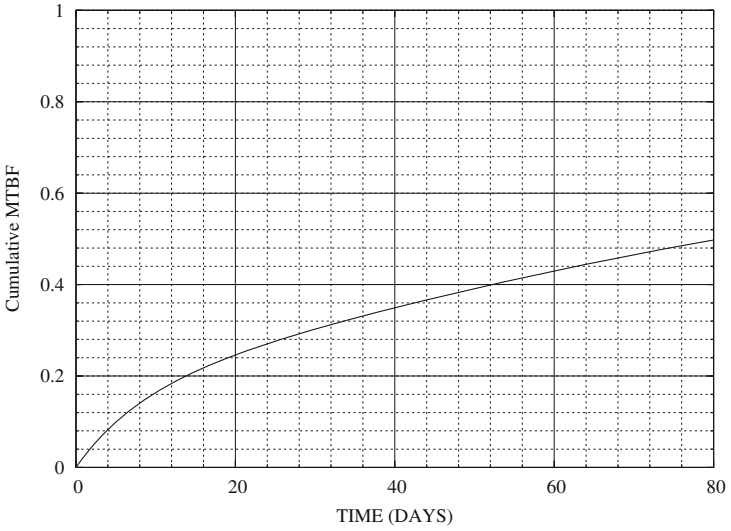

Fig. 11.14 The estimated MTBF_C, $\widehat{MTBF}_C(t)$

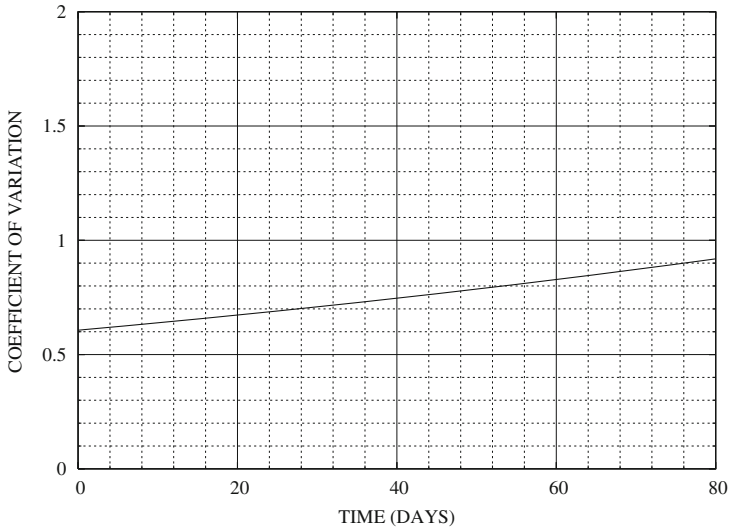

Fig. 11.15 The estimated coefficient of variation, $CV(t)$

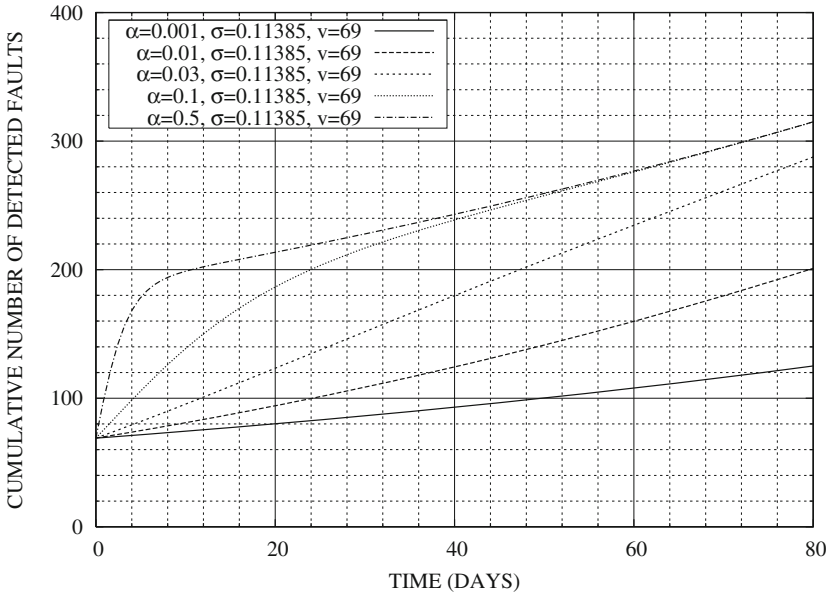

Fig. 11.16 Dependence of model parameter α

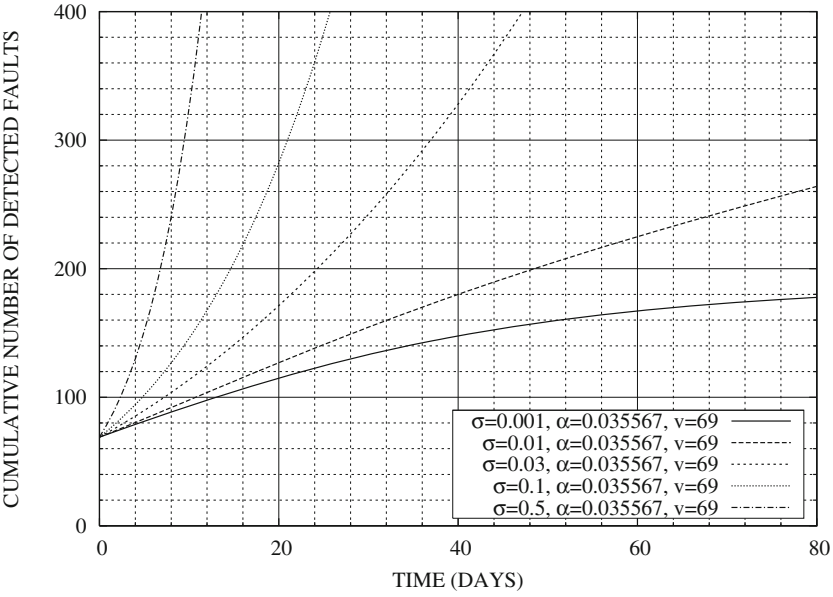

Fig. 11.17 Dependence of model parameter σ

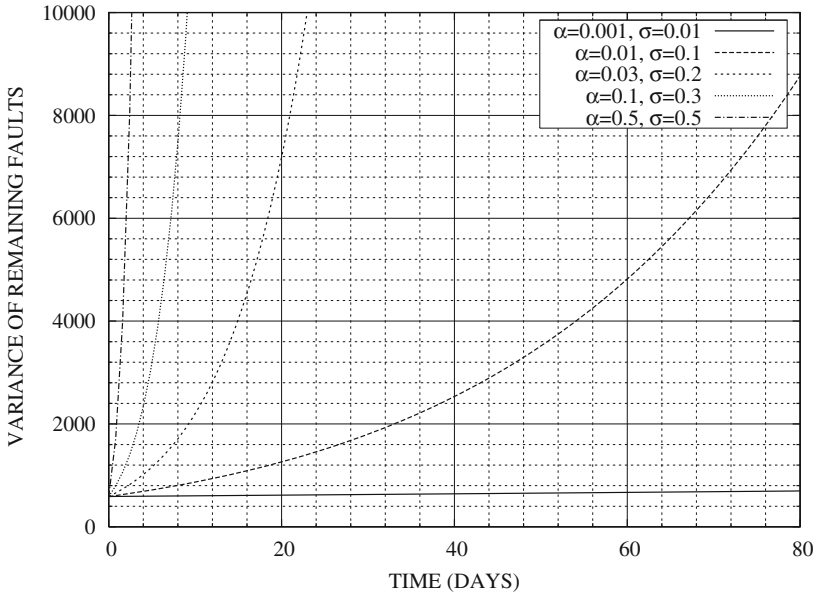

Fig. 11.18 Dependence of model parameters for the variance of the number of detected faults

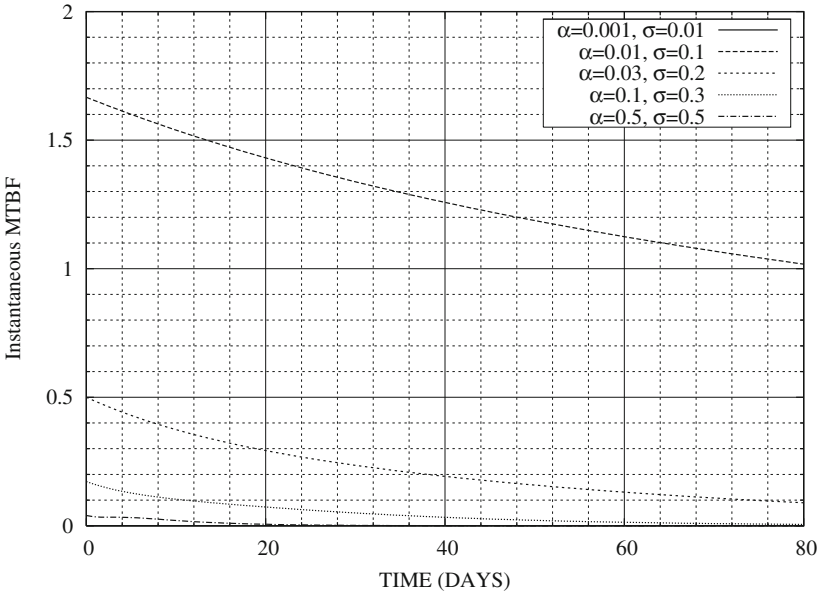

Fig. 11.19 Dependence of model parameter for the $MTBF_I$

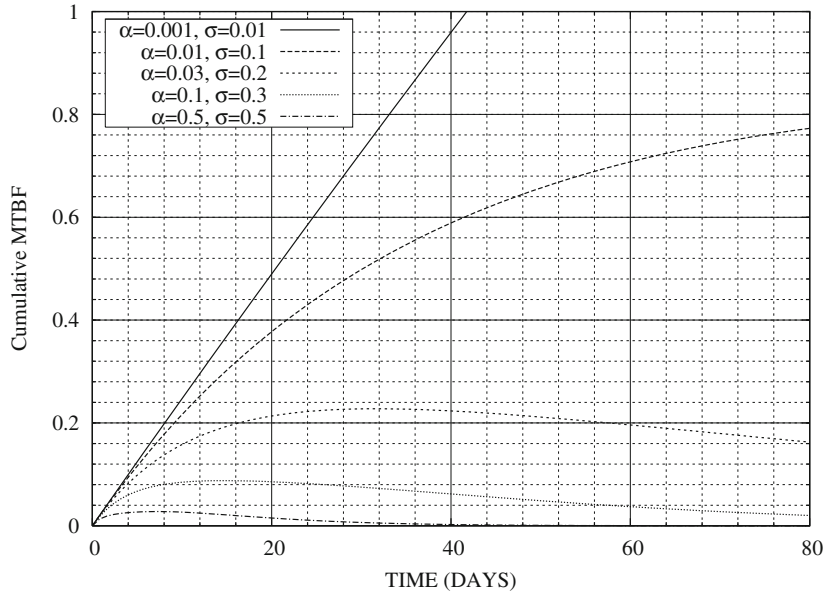

Fig. 11.20 Dependence of model parameter for $MTBF_C$

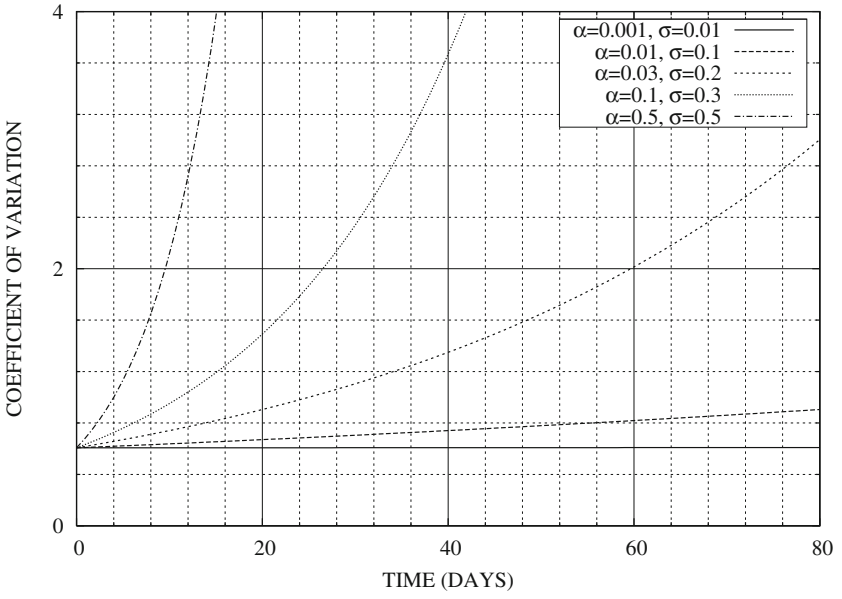

Fig. 11.21 Dependence of model parameter for the coefficient of variation

Table 11.14 The actual data in Apache HTTP Server

Unit time (Days)	Cumulative number of detected faults	Unit time (Days)	Cumulative number of detected faults
0	9	29	37
1	11	30	38
2	11	31	39
3	12	32	39
4	15	33	39
5	17	34	41
6	18	35	41
7	18	36	41
8	18	37	42
9	20	38	42
10	21	39	43
11	24	40	45
12	24	41	45
13	25	42	45
14	26	43	47
15	26	44	49
16	26	45	49
17	27	46	49
18	28	47	50
19	31	48	51
20	31	49	51
21	31	50	53
22	32	51	53
23	32	52	53
24	32	53	55
25	32	54	57
26	33	55	59
27	35	56	60
28	35		

HTTP Server Project[2] have developed and maintained an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. Table 11.14 shows actual data sets used in this chapter.

[2]The Apache HTTP Server is a project of the Apache Software Foundation.

The sample path of the estimated numbers of detected faults in our model and existing model [4], $\widehat{S}(t)$, are shown in Figs. 11.22 and 11.23, respectively. From Figs. 11.22 and 11.23, we have found that the magnitude of the irregular fluctuation in Fig. 11.22 is larger than the one in Fig. 11.23, i.e., the software failure intensity in our model depends on the time.

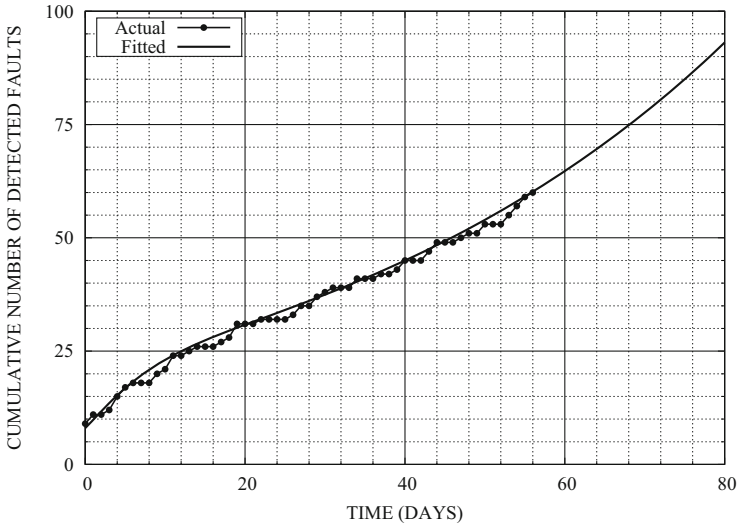

Fig. 11.22 The estimated number of detected faults in our model

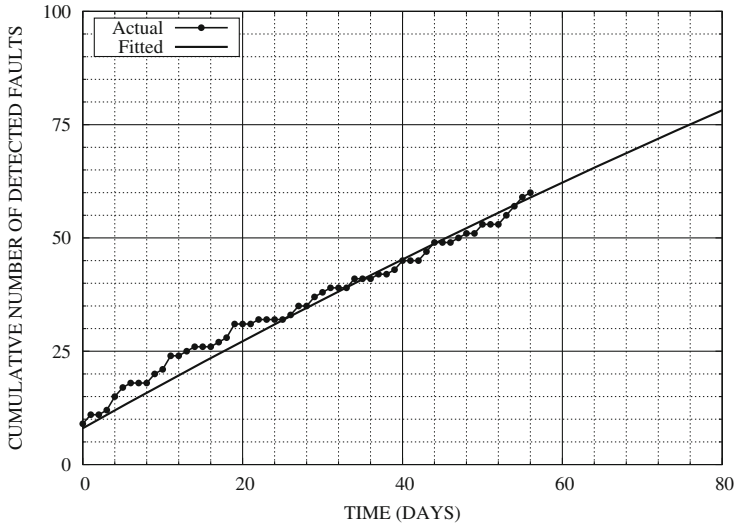

Fig. 11.23 The estimated number of detected faults in existing model

Table 11.15 Comparison of the mean squared errors for the cumulative number of detected faults

Compared models	MSE
Our model	1.5447
Conventional model	4.1234

Moreover, we compare the goodness-of-fit of our model with the existing model [4] based on stochastic differential equations for the observed data set. We adopt the value of the Mean Square Error (MSE) as comparison criteria of goodness-of-fit. Suppose that K data pairs $(t_k, y_k)(k = 1, 2, \ldots, K)$ are observed during the operation phase where y_k is the cumulative number of software faults observed in the time-interval $(0, t_k]$. MSE can be obtained by dividing the sum of square errors between the observed value, y_k, and the estimated one, $\hat{y}_k$, by the number of data pairs, K. That is,

$$\mathrm{MSE} = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2, \tag{11.1}$$

where $\hat{y}_k$ in Eq. (11.1) is obtained from estimated expected values. MSE indicates that the selected model fits better to the observed data as the MSE becomes small.

Table 11.15 shows the result of goodness-of-fit comparison in terms of the MSE for our model and the existing model based on stochastic differential equations. We find that our model fits better than the existing model with respect to MSE.

11.3.5 Discussion for the Method of Reliability Assessment Based on Stochastic Differential Equation Models

We have focused on the Fedora Core Linux operating system and the Apache HTTP Server software which are known as the OSS, and discussed the method of reliability assessment for the OSS developed under an open source project.

Moreover, we have proposed a software reliability growth model based on stochastic differential equations in order to consider the active state of the open source project. Especially, we have assumed that the software failure intensity depends on the time, and the software fault-report phenomena on the bug tracking system keep an irregular state. Also, we have analyzed actual software fault-count data to show numerical examples of software reliability assessment for the OSS. Moreover, we have compared our model with the conventional model based on stochastic differential equations in terms of goodness-of-fit for actual data.

Finally, we have focused on an OSS developed under open source projects. New distributed development paradigm typified by such open source project will evolve at a rapid pace in the future. Our method is useful as the method of reliability assessment after the release of the evaluation version of OSS.

11.4 Hazard Rates for Embedded OSS

11.4.1 Embedded OSS

There are many open source projects around the world. In particular, we focus on the embedded OSS in order to evaluate the performance of our method, i.e., Android [5] and BusyBox [6]. BusyBox includes 4 components. We show numerical examples by using the data after "Android 1.5 NDK, Release" and "BusyBox 1.10.1 (stable)" as Figs. 11.24 and 11.25.

In this section, we focus on the Android OS developed for mobile phone. In particular, we consider the case of installing BusyBox to Android as the porting environment. Thus, we illustrate the method of reliability assessment for the porting-phase.

11.4.2 Reliability Assessment

We analyze the actual data in terms of software failure-occurrence time-interval. Based on our flexible hazard rate model, the following model parameters have been estimated:

$$\hat{w_1} = 0.49840,\ \hat{w_2} = 0.13419,$$
$$\hat{\alpha} = 0.024757,\ \hat{N} = 99.4456,$$

where we assume as $w_1 = pD$ and $w_2 = (1-p)\phi$ for the simplification technique. Moreover, we define w_1 and w_2 as the scale parameters of the porting stability. We find the degree of maturation of Android OS means the low level from the estimation results of our model parameters. Thus, the degree of maturation of OSS means the low level, if w_1 is large. On the other hand, it means the high level, if w_2 is large.

First, Fig. 11.26 shows the behavior of the estimated MTBF for Android in Eq. (8.2). From Fig. 11.26, we find that the proposed hazard rate model for embedded OSS fits better than the conventional Schick-Wolverton [7] model with respect to MTBF.

Next, the estimated MTBF in Eq. (8.3) is shown in Fig. 11.27. Using the BusyBox data, the estimated hard rate model in Fig. 11.27 is applied for the specific software component in this chapter.

Moreover, we show the estimated MTBF in Eq. (8.1) in case of installing Android onto BusyBox in Fig. 11.28. From Fig. 11.28, we can confirm that the MTBF grows as porting procedures go on. Also, the estimated software reliability $R_{30}(x)$ is shown in Fig. 11.29.

ID	Type	Status	Summary	Stars	Component	Reporter	Opened	Priority
15	Enhancement	Reviewed	no proper multicast support in emulator	19	Tools	androiddev001	2007.11.22 21:56	Medium
41	Enhancement	Reviewed	feature request for Eclipse Source->XML-> externalize/internalize plugin	6	Tools	jehrl...@dairyvets.com	2008.1.19 19:03	Medium
43	Defect	New	Eclipse shows warnings for R.java	2	Tools	rich...@kennardconsulting.com	2008.1.19 20:17	Medium
53	Enhancement	Reviewed	Can't get to know the phone number of an incoming call	22	Applications	hongbinm	2008.1.21 2:38	Medium
54	Enhancement	Reviewed	Can't reject a call	36	Framework	hongbinm	2008.1.21 2:38	Medium
67	Enhancement	Reviewed	missing access to Audio samples (feature request)	20	GfxMedia	JogiNeumann	2008.1.22 11:32	Medium
78	Enhancement	Reviewed	Cant load PDF data	12	Applications	Paolo.G.Romano	2008.1.24 10:39	Medium
82	Enhancement	Reviewed	wifi - support ad hoc networking	51	System	wink.to.teleca	2008.1.24 20:03	Medium
83	Enhancement	Reviewed	Allow apps to supply alternate components to the media streaming subsystem.	26	GfxMedia	wink.to.teleca	2008.1.24 20:21	Medium
85	Enhancement	Reviewed	Need ability to fiddle with apk files	2		davidnwelton	2008.1.24 23:54	Medium
86	Enhancement	Reviewed	Extend contacts to allow better third party interoperability	21	Applications	aplocher	2008.1.25 5:02	Medium
126	Enhancement	Reviewed	No way to figure out the number dialed in an outbound call	4	Framework	asafxrev	2008.1.29 16:42	Medium
145	Enhancement	Reviewed	No way to track player progress of audio clips using MediaPlayer	5	Framework	charlie.hubbard	2008.1.31 2:39	Medium
152	Enhancement	Reviewed	Taking Screenshot from within service	10	GfxMedia	kerstenk	2008.1.31 20:46	Medium
159	Enhancement	Reviewed	Wrap ContentProvider boilerplate logic using dynamic XML schema	6	Framework	luke.hutch	2008.2.2 1:08	Medium
178	Enhancement	New	Export of classes (Enhancement)	2		marcus.mitschack	2008.2.3 17:05	Medium
212	Defect	Reviewed	Documented People.query method does not exist	2		tcstellanova	2008.2.8 18:32	Medium
215	Defect	Reviewed	Documentation: too much copy&paste in R.styleable document	1		michal.selmi.seliga	2008.2.8 23:48	Medium
227	Enhancement	Reviewed	Should be able to update a specific list item	4	Framework	vijay.r.shankar	2008.2.11 7:41	Medium
229	Enhancement	Reviewed	Feature request: Direct allocation of a bitmap's underlying buffer in BitmapFactory	3		laurent.pontier	2008.2.11 13:24	Medium
235	Enhancement	Reviewed	Telephonestate ONHOOK	3		kerstenk	2008.2.12 14:56	Medium
236	Enhancement	Reviewed	Telephonestate call active	3		kerstenk	2008.2.12 14:59	Medium
263	Defect	NeedsInfo	SurfaceView punch hole in Absolute Layout with Animation..	1		sungsuh.park	2008.2.15 12:29	Medium
265	Defect	Reviewed	TabHost-Example within its Javadoc is outdated.	1		NicolasGramlich	2008.2.15 15:15	Medium
268	Enhancement	Reviewed	Use Java annotations to automatically pack/unpack properties to/from icicle	2		luke.hutch	2008.2.15 16:34	Medium
274	Defect	Reviewed	onRestart and onStart documentation contradictory	3		joe.bowbeer	2008.2.16 3:54	Medium
280	Enhancement	Reviewed	PhoneStateIntentReceiver forces design that takes more system resources	3		asafxrev	2008.2.16 19:20	Medium
294	Defect	NeedsInfo	Drawable.createFromStream() and Drawable.createFromPath() issue	1		cooljalopy	2008.2.18 8:13	Medium
301	Enhancement	Reviewed	MJPG streaming from Axis Cameras does not work	3		ethanet	2008.2.18 15:50	Medium

Fig. 11.24 The partial source data in Android

Id	Project	Reporter	Assigned	To	Priority	Severity	Product	Version	Projection	Category	Version	Platform
2664	buildroot	jbruchon	buildroot	normal	minor	have	tried		none	Shared		
2674	buildroot	antab	buildroot	normal	major	always	none	Standards	Compliance	03-25-08	03-25-08	patch-kernel.sh
2644	buildroot	user3244	buildroot	normal	minor	always	none	Architecture	Specific	03-23-08	03-26-08	incorrect
2654	buildroot	jbruchon	buildroot	normal	minor	always	none	Other	03-23-08	none	Snapshot	downloads
2634	buildroot	travishein	buildroot	normal	minor	always	none	Other	03-22-08	none	path	to
2684	buildroot	Graol	buildroot	normal	block	always	none	Architecture	Specific	03-26-08	03-27-08	Fakeroot
2694	buildroot	ychuang	buildroot	normal	major	always	none	Architecture	Specific	03-26-08	03-28-08	busybox
2744	BusyBox	ianw	BusyBox	normal	minor	always	none	Other	2004.2.8 0:00	none	cmp_common	should
2754	buildroot	andrewma	buildroot	normal	tweak	always	none	Architecture	Specific	2004.2.8 0:00	2004.3.8 0:00	make
2774	buildroot	davidg	buildroot	normal	minor	always	none	Architecture	Specific	2004.3.8 0:00	2004.3.8 0:00	liblockfile
2794	buildroot	davidg	buildroot	normal	minor	always	none	Architecture	Specific	2004.3.8 0:00	2004.3.8 0:00	build
2764	BusyBox	Peter	Eisentraut	BusyBox	normal	feature	not	tried		none		
2804	buildroot	rah	buildroot	normal	minor	always	none	Architecture	Specific	2004.3.8 0:00	2004.4.8 0:00	grub_0.97-31.diff
2824	buildroot	fabo	buildroot	normal	minor	always	none	Other	2004.4.8 0:00	none	gdb	2006.7.1 0:00
2854	buildroot	Andrew	Mahone	buildroot	normal	tweak	2000.9.27 0:00	none	Other	2004.5.8 0:00	2004.5.8 0:00	zlib.mk
2844	buildroot	Andrew	Mahone	buildroot	normal	tweak	2000.9.27 0:00	none	Other	2004.5.8 0:00	2004.5.8 0:00	bash.mk
2874	buildroot	joedawson	buildroot	normal	minor	always	none	Architecture	Specific	2004.7.8 0:00	2004.7.8 0:00	libglib2
2714	uClibc	vda	uClibc	normal	minor	always	none	Stdio	2004.1.8 0:00	none	sprintf	should
2924	BusyBox	kiltedknight	BusyBox	normal	feature	always	none	New	Features	04-14-08	04-14-08	ash
2934	uClibc	creber	uClibc	normal	minor	always	none	Architecture	Specific	04-16-08	04-16-08	[ARM]
2894	uClibc	woglinde	khem	normal	minor	always	none	Architecture	Specific	2004.11.8 0:00	04-19-08	extra/scripts/gen_bits_syscall_h.sh
2954	buildroot	Jean-Baptiste	Maillet	buildroot	normal	minor	2000.9.27 0:00	none	Other	04-20-08	04-20-08	package cgicc no longer available from aut
2964	buildroot	Jean-Baptiste	Maillet	buildroot	normal	minor	2000.9.27 0:00	none	Other	04-20-08	04-20-08	package libosip2, source and build fails
2984	buildroot	maluta	buildroot	normal	minor	always	none	Architecture	Specific	04-20-08	04-20-08	[error]
2994	BusyBox	thorstenhirsch	BusyBox	normal	major	always	none	Networking	Support	04-20-08	04-20-08	SIOCBRADDBR
3014	BusyBox	wenjinshan	BusyBox	normal	minor	always	none	Kernel	Module	Support	public	04-21-08
3044	buildroot	gebecom	buildroot	normal	minor	always	none	Architecture	Specific	04-22-08	04-22-08	problem:asterisk
3054	buildroot	gebecom	buildroot	normal	minor	always	none	Architecture	Specific	04-22-08	04-22-08	error
3004	buildroot	gebecom	buildroot	normal	minor	always	none	Architecture	Specific	04-21-08	04-22-08	undefined

Fig. 11.25 The partial source data in BusyBox

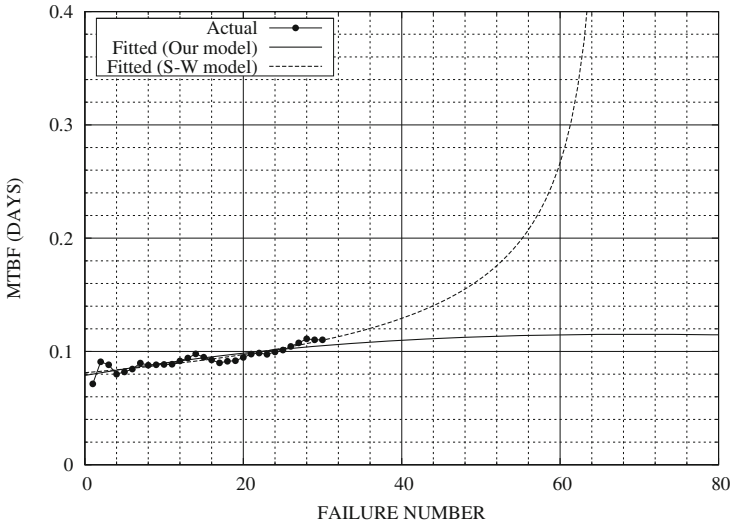

Fig. 11.26 The estimated MTBF in Android (Up to 30 days of data)

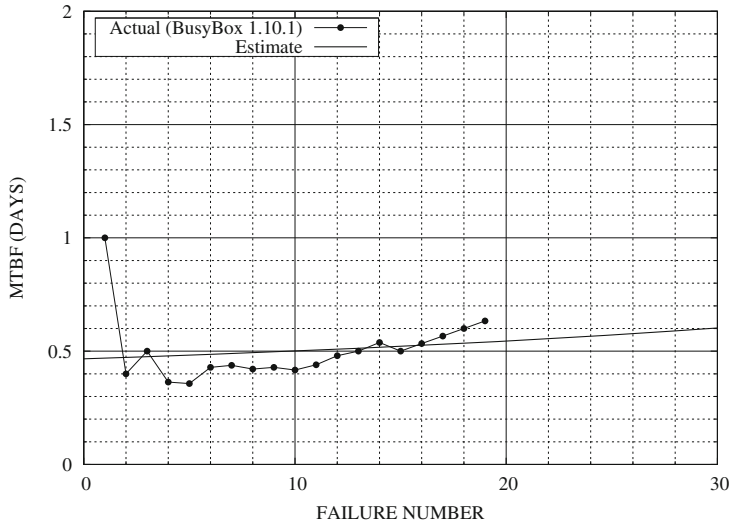

Fig. 11.27 The estimated MTBF in BusyBox

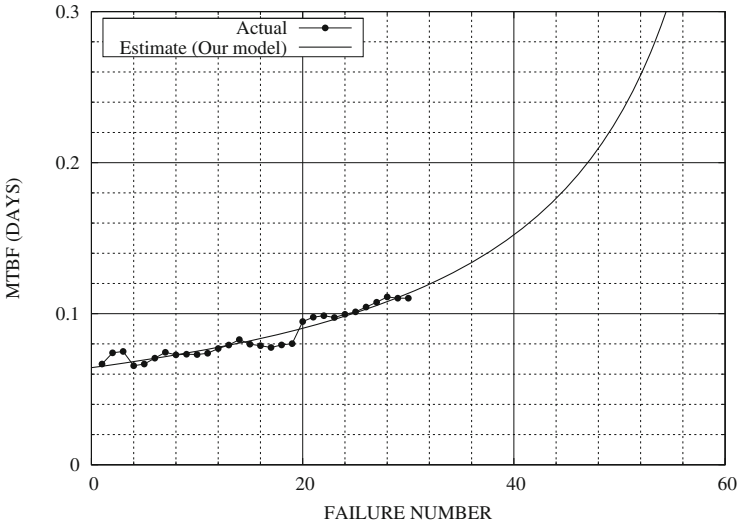

Fig. 11.28 The estimated MTBF in case of installing Android onto BusyBox (Up to 30 days of data)

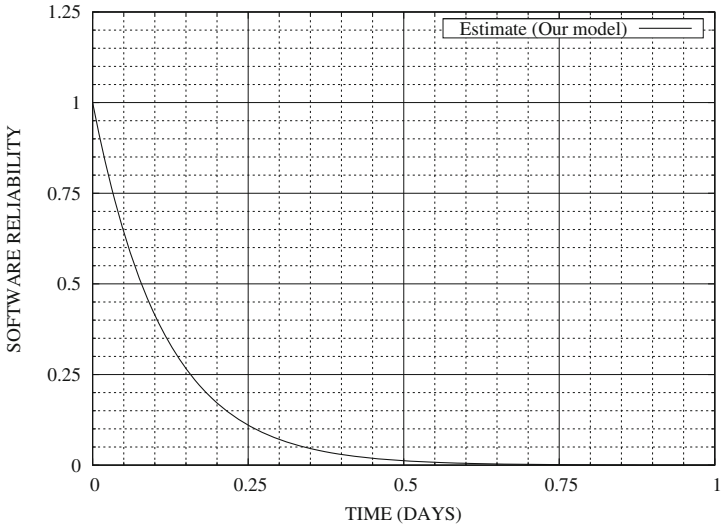

Fig. 11.29 The estimated software reliability

11.4.3 Comparison of Goodness-of-Fit

11.4.3.1 Compared Hazard Rate Models

We compare our flexible hazard rate model for embedded OSS with the following typical conventional hazard rate models:

(for Schick–Wolverton(S-W) model)

$$z_k(x) = \phi(N-k+1)x \quad (N>0,\ \phi>0;\ k=1,\ 2,\ldots,N), \tag{11.2}$$

$$\mathrm{E}[X_k] = \sqrt{\frac{\phi}{2(N-k+1)\phi}}. \tag{11.3}$$

(for Jelinski–Moranda(J-M) model)

$$z_k(x) = \phi(N-k+1) \quad (N>0,\ \phi>0;\ k=1,\ 2,\ldots,N), \tag{11.4}$$

$$\mathrm{E}[X_k] = \frac{1}{\phi(N-k+1)}. \tag{11.5}$$

(for Moranda model)

$$z_k(x) = Dc^{k-1} \quad (D>0,\ 0<c<1;\ k=1,\ 2,\ldots), \tag{11.6}$$

$$\mathrm{E}[X_k] = \frac{1}{Dc^{k-1}}. \tag{11.7}$$

(for Xie model)

$$z_k(x) = \lambda_0(N-k+1)^{\alpha} \quad (N>0,\ \lambda_0>0,\ \alpha\geq 1;\ k=1,\ 2,\ldots,N), \tag{11.8}$$

$$\mathrm{E}[X_k] = \frac{1}{\lambda_0(N-k+1)^{\alpha}}. \tag{11.9}$$

The model parameters in Eqs. (11.2)–(11.9) are defined as follows:

- N the latent fault in software system,
- ϕ the hazard rate per remaining fault,
- D the initial hazard rate for 1st software failure,
- c the reduction factor of hazard rate,
- λ_0 the hazard rate per remaining fault considering α,
- α the constant parameter.

11.4.3.2 Comparison Criteria of Goodness-of-Fit

We adopt the Mean Square Error (MSE) as a comparison criterion of goodness-of-fit.

MSE can be obtained from dividing the sum of square errors between the observed value, y_k, and the estimated one, $\hat{y}_k$, by the number of data pairs, K. That is,

$$\mathrm{MSE} = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2, \tag{11.10}$$

where $\hat{y}_k$ in Eq. (11.10) is obtained from estimated $\mathrm{E}[X_k]$, $(k = 1, 2, \ldots, K)$ for each hazard rate model. The MSE indicates that the selected models fit better to the observed data as the MSE becomes small.

11.4.3.3 Performance Evaluation of Our Model

Table 11.16 shows actual data sets.

We compare the goodness-of-fit of the our hazard rate model with that of the conventional S-W model [7], J-M model [8], Moranda model [9], and Xie model [10] after the end of fault-detection report up to 30 days. The estimated mean time between software failures of our model, S-W model [7], J-M model [8], Moranda model [9], and Xie model [10] are shown in Fig. 11.30, respectively. From Fig. 11.30, we can confirm that the our hazard rate model for embedded OSS fits better than the conventional hazard rate models with respect to MTBF after the end of fault-detection report up to 30 days.

11.4.4 Prediction Accuracy After the End of Fault-Report

We compare the goodness-of-fit of the hazard rate models after the end of fault-detection report.

We analyze the prediction accuracy based on the data from the beginning of porting-phase of embedded system to the end of fault-detection report in embedded OSS. Table 11.17 shows the comparison results of the MSE for the mean time between software failures in terms of the compared models. The values of MSE in Table 11.17 represent the estimate until the end of 30 days and 50 days.

From Table 11.17, we find that our model is rarely different from the conventional hazard rate models in terms of the value of MSE until 30 days. In particular, our model fits better than the conventional hazard rate models in terms of the value of MSE until the end of 50 days. Moreover, we can confirm that the our model fits better for the long-term prediction accuracy. Thereby, we can conduct effective software reliability prediction for the porting-phase of embedded system development.

Table 11.16 The actual MTBF data in Android and BusyBox

Failure number	MTBF (Days)	Failure number	MTBF (Days)
1	0.06667	26	0.10442
2	0.07407	27	0.10757
3	0.07500	28	0.11111
4	0.06557	29	0.11027
5	0.06667	30	0.11029
6	0.07059	31	0.10839
7	0.07447	32	0.11034
8	0.07273	33	0.11111
9	0.07317	34	0.11371
10	0.07299	35	0.11589
11	0.07383	36	0.11538
12	0.07692	37	0.11563
13	0.07927	38	0.11656
s14	0.08284	39	0.11642
15	0.07979	40	0.11730
16	0.07882	41	0.11816
17	0.07763	42	0.11966
18	0.07930	43	0.11944
19	0.08017	44	0.11796
20	0.09479	45	0.11780
21	0.09767	46	0.11795
22	0.09865	47	0.11839
23	0.09746	48	0.11707
24	0.09959	49	0.11779
25	0.10121	50	0.11765

11.4.5 Optimal Software Release Problem for the Porting-Phase of Embedded OSS

11.4.5.1 Formulation of Total Software Cost

Recently, it becomes more difficult for software developers to produce highly-reliable software systems efficiently, because of the more diversified and complicated software requirements. Thus, it has been necessary to control the software development process in terms of reliability, cost, and delivery time [11, 12]. Especially, it is difficult for software developers to manage the porting-phase of the embedded system development using the embedded OSS. Also, it is very important in terms of software management that we decide for the optimal length of the porting-phase for embedded OSS. We find the optimal release time of porting-phase by minimizing the total

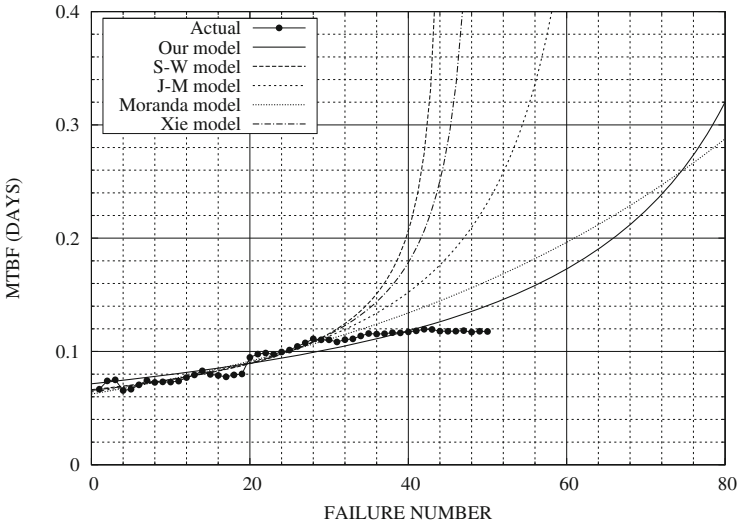

Fig. 11.30 The estimated MTBF for each hazard rate models

Table 11.17 Comparison of the MSE for the estimated MTBF

	MSE (30 days)	MSE (50 days)
Our model	4.8467×10^{-5}	6.2665×10^{-5}
S-W model	1.9268×10^{-5}	0.007146
J-M model	1.9545×10^{-5}	0.001010
Moranda model	2.2923×10^{-5}	0.0002023
Xie model	0.007381	0.04178

expected software maintenance cost in this section. We formulate a maintenance cost model based on our proposed hazard rate model for the embedded OSS. It is interesting for the software developers to predict and estimate the time when we should stop bug fixing in order to develop a highly reliable software system efficiently. Then, we discuss about the determination of optimal software release times minimizing the total expected software cost. We define the following:

c_1 the testing cost per porting-time ($c_1 > 0$),
c_2 the fixing cost per fault during the porting-phase ($c_2 > 0$),
c_3 the fixing cost per fault after the release ($c_3 > c_2$).

Then, the expected software cost of OSS can be formulated as:

$$C_1(l) = c_1 \sum_{k=1}^{l} \mathrm{E}[X_k] + c_2 l, \tag{11.11}$$

where l is the number of software failure-occurrence.

Also, we can define the expected software maintenance cost for ith software component as follows:

$$C_2^i(l) = c_3 (N - l) . \tag{11.12}$$

Consequently, from Eqs. (11.11) and (11.12), the total expected software cost is given by

$$C(l) = C_1(l) + C_2(l). \tag{11.13}$$

From l^* obtained by minimizing l, we can estimate the optimum software release time $\sum_{k=1}^{l^*} \mathrm{E}[X_k]$.

11.4.5.2 Numerical Illustration of Optimal Software Release Time

We show numerical examples of reliability assessment for Android [5] and BusyBox [6]. Figure 11.31 shows the estimated total expected software cost where $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$. We assume that c_1, c_2, and c_3 are estimated by the software managers. From Fig. 11.31, we find that the estimated number of failure-occurrence, l^*, which minimizes the estimated total expected software cost is 96. Then, the optimal software release time $t^* = \sum_{k=1}^{l^*} \mathrm{E}[X_k]$ is 20.473 days. From the result, the total expected software cost $C(l^*)$ is 222.5.

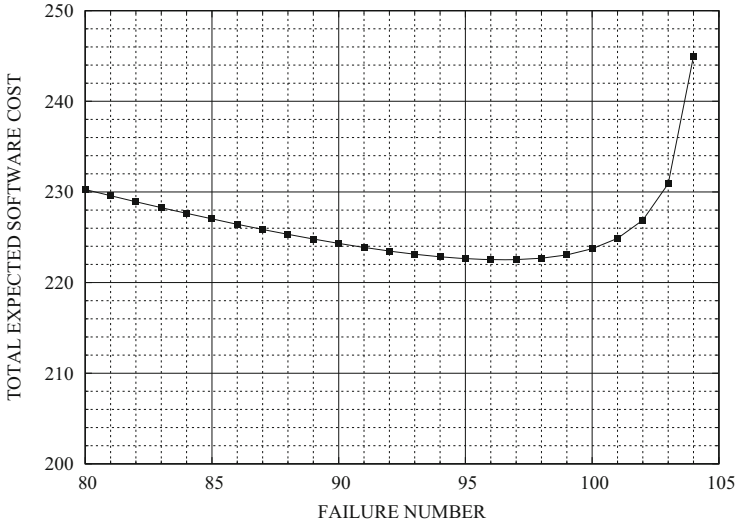

Fig. 11.31 The estimated total expected software cost

11.4.5.3 Optimal Release Problem with Reliability of Embedded OSS

Also, we consider that the developer of embedded system can estimate the optimal release time with considering reliability by combining the total expected software cost with the reliability in the Sect. 5.3. We assume that the reliability requirement as follows:

$$R_k(1) = 0.7.$$

We obtain the reliability $R_{101}(1) = 0.7$ when the optimum release time is $t'^* = 31.191$ days where $l'^* = 101$. Then, we need to lengthen the porting-time, which is shown in Fig. 11.32 as $(t'^* - t^*) = 31.191 - 20.473 = 10.718$ days. Figure 11.32 illustrates the optimum release time with reliability objective $R_{101}(1) = 0.7$.

From Fig. 11.32, we have found that the optimum release time become lengthen, and the total expected software cost increases. When the porting time and the reliability objective are assumed as $R_k(1) = 0.7$, we obtain $t'^* = 31.191$ from $l^* = 101$. Then, the total expected software cost is 224.9.

11.4.6 Discussion for the Method of Reliability Assessment Based on Hazard Rates for Embedded OSS

We have discussed the method of software reliability assessment based on flexible hazard rate model for embedded OSS. In particular, we have derived several assessment measures based on our hazard rate model. By using our flexible hazard rate model, we can incorporate the complicated situation of the embedded system used OSS, the degree of maturation of OSS, uniquely software components such as device driver, etc. In case of considering the effect of debugging process on software reliability assessment for open source projects, it is necessary to grasp the deeply-intertwined factors. In this section, we have shown that our model can describe such deeply-intertwined factors.

Moreover, we have compared the goodness-of-fit of our model discussed in this chapter with the conventional hazard rate models. Then, we have shown that the proposed model can assist improvement of quality for embedded OSS systems development. Thereby, our hazard rate model will reduce some efforts to select a suitable model for the collected data sets.

Furthermore, we have formulated the total expected software cost based on our hazard rate model. We have found that our method can evaluate the optimum software release time in the porting-phase of the embedded system development by applying embedded OSS.

By using our method, the embedded software developer can conduct an effective software reliability prediction for the porting-phase of embedded system development.

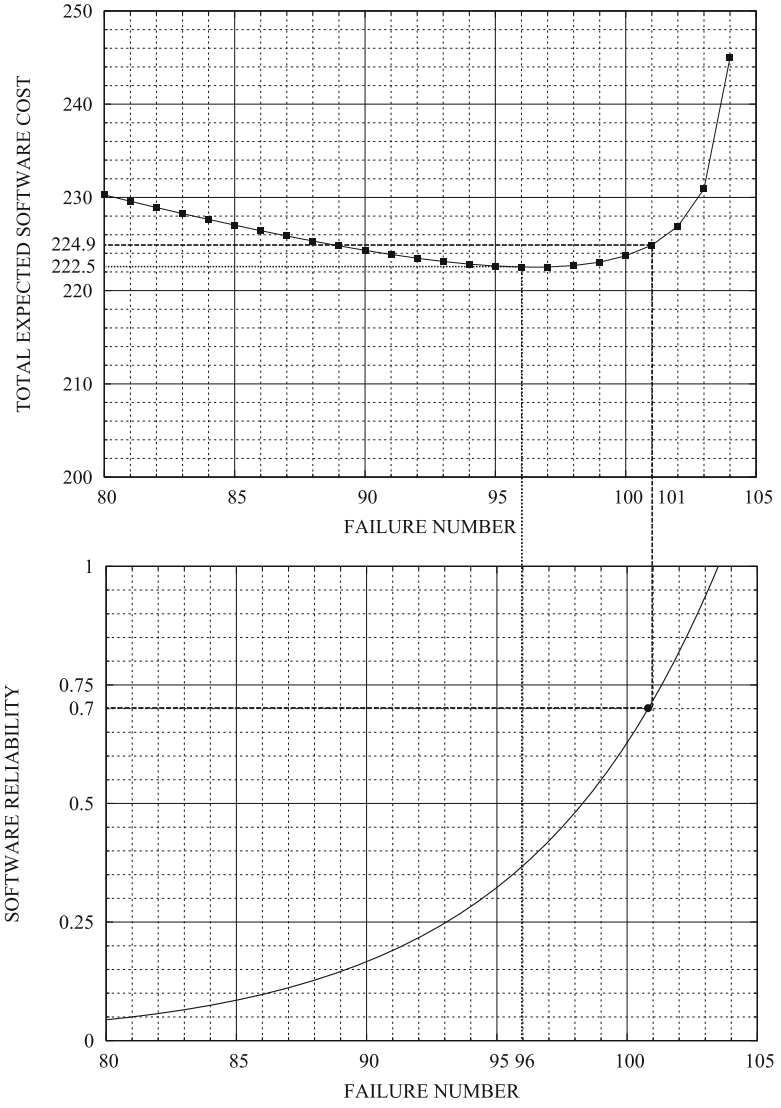

Fig. 11.32 The optimum software release time with reliability requirement, t'^{*}

11.5 Applied Examples for Open Source Solution

11.5.1 Data for Numerical Illustrations

We focus on a large-scale open source solution based on the Apache HTTP Server [3], Apache Tomcat [13], MySQL [14] and JSP (JavaServer Pages). The fault-count

Table 11.18 The actual data in open source solution

Time (Weeks)	Cumulative number of detected faults
0	24
1	27
2	31
3	32
4	37
5	38
6	39
7	42
8	44
9	46
10	48
11	50

data used in this section are collected in the bug tracking system on the website of each open source project. Table 11.18 shows the actual data set.

11.5.2 Reliability Assessment

The estimated expected cumulative numbers of detected faults in Eq. (9.19), $\widehat{\mathrm{E}}[S(t)]$'s, in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$ are shown in Figs. 11.33 and 11.34, respectively. Also, the sample paths of the estimated numbers of detected faults in Eq. (9.7), $\widehat{S}(t)$'s, in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$ are shown in Figs. 11.35 and 11.36, approximately. Moreover, the estimated expected cumulative numbers of remaining faults in Eq. (9.22), $\widehat{\mathrm{E}}[N(t)]$'s, in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$ are shown in Figs. 11.37 and 11.38, respectively.

Furthermore, the estimated $\mathrm{MTBF_C}$'s in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$ are plotted in Figs. 11.39 and 11.40, respectively. These figures show that the $\mathrm{MTBF_C}$ increase as the testing procedures go on. Also, $CV(t)$'s in case of $\lambda(t) = \lambda_1(t)$ and $\lambda(t) = \lambda_2(t)$ are shown in Figs. 11.41 and 11.42, respectively. From Figs. 11.41 and 11.42, we can confirm that the estimated coefficient of variation approaches the constant value. We consider that the coefficient of variation is useful measure to compare several fault data sets in the past system development projects.

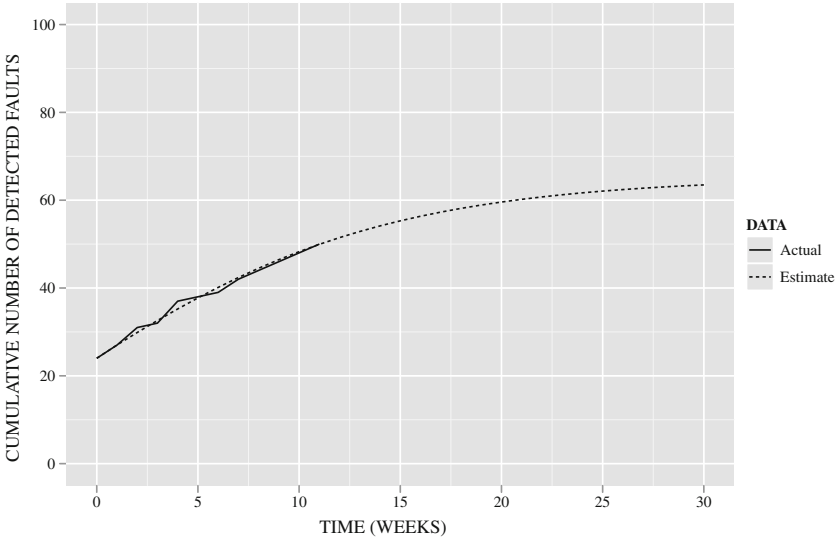

Fig. 11.33 The estimated cumulative number of detected faults, $\hat{\mathrm{E}}[S(t)]$, in case of $\lambda(t) = \lambda_1(t)$

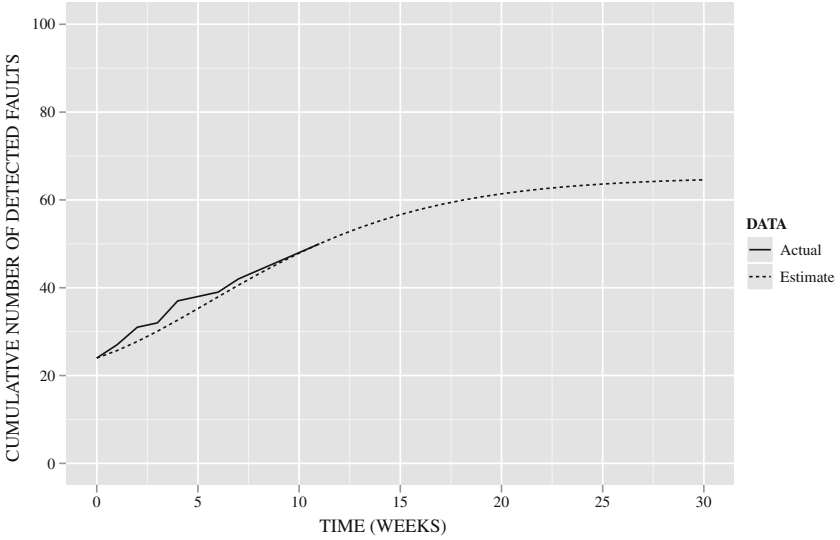

Fig. 11.34 The estimated cumulative number of detected faults, $\hat{\mathrm{E}}[S(t)]$, in case of $\lambda(t) = \lambda_2(t)$

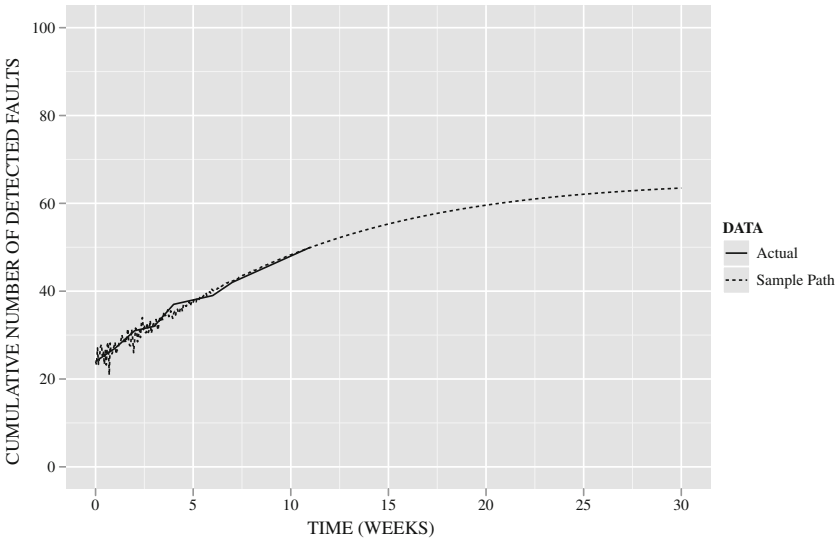

Fig. 11.35 The sample path of the estimated number of detected faults, $\widehat{S}(t)$, in case of $\lambda(t) = \lambda_1(t)$

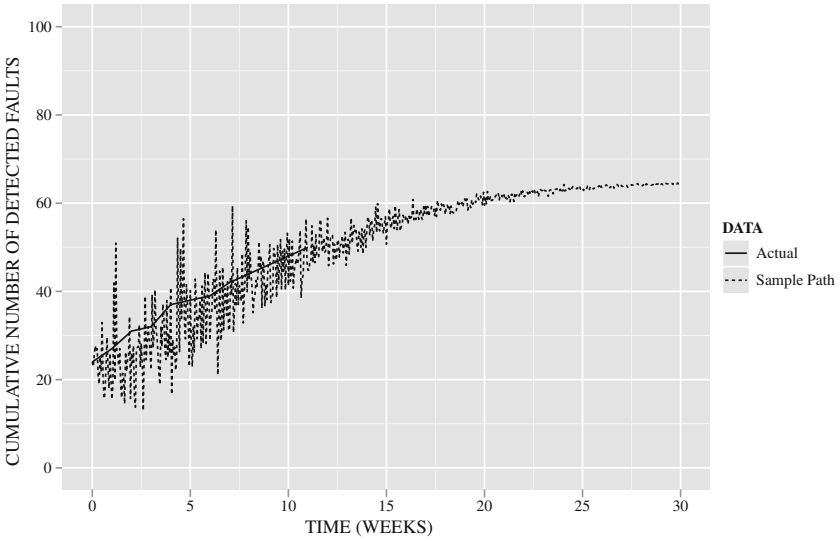

Fig. 11.36 The sample path of the estimated number of detected faults, $\widehat{S}(t)$, in case of $\lambda(t) = \lambda_2(t)$

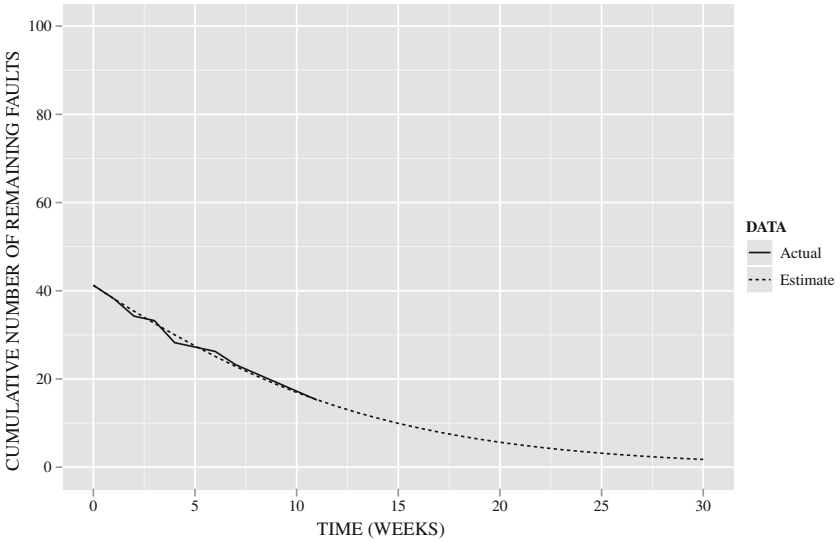

Fig. 11.37 The estimated cumulative number of remaining faults, $\hat{\mathrm{E}}[S(t)]$, in case of $\lambda(t) = \lambda_1(t)$

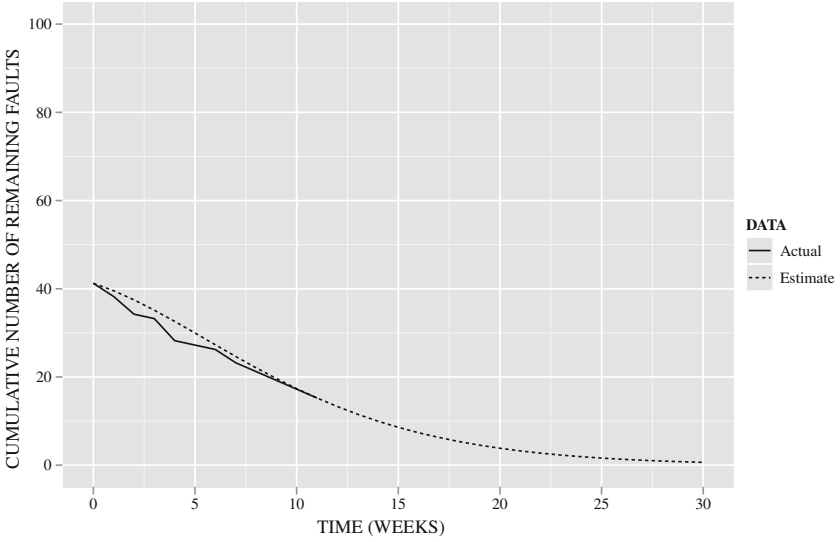

Fig. 11.38 The estimated cumulative number of remaining faults, $\hat{\mathrm{E}}[S(t)]$, in case of $\lambda(t) = \lambda_2(t)$

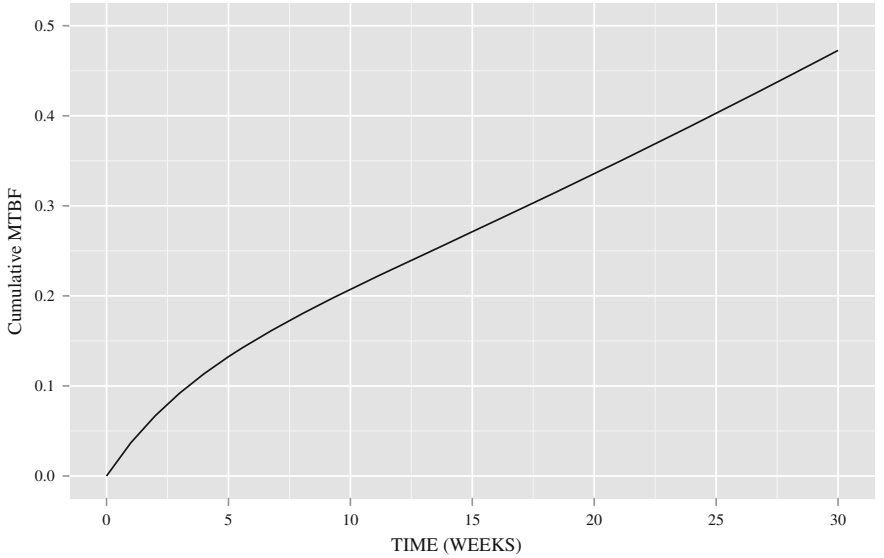

Fig. 11.39 The estimated MTBF$_C$, $\widehat{MTBF}_C(t)$, in case of $\lambda(t) = \lambda_1(t)$

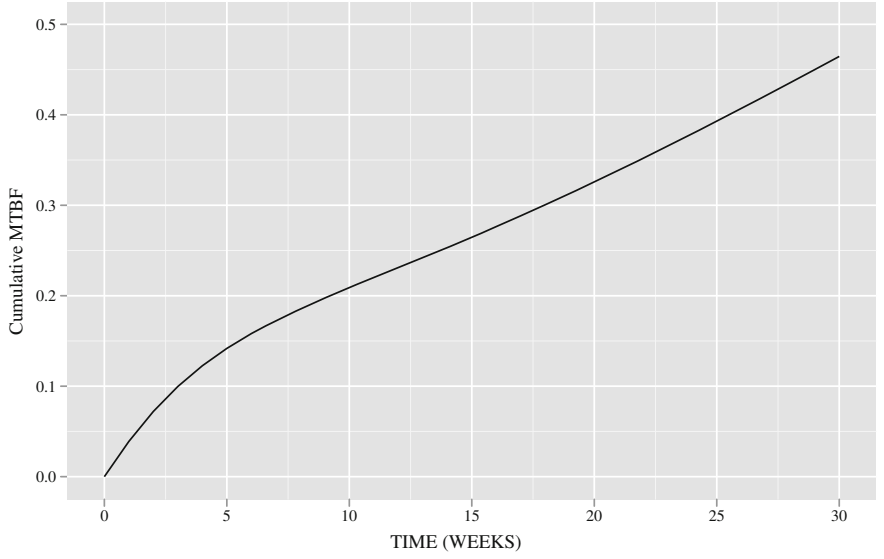

Fig. 11.40 The estimated MTBF$_C$, $\widehat{MTBF}_C(t)$, in case of $\lambda(t) = \lambda_2(t)$

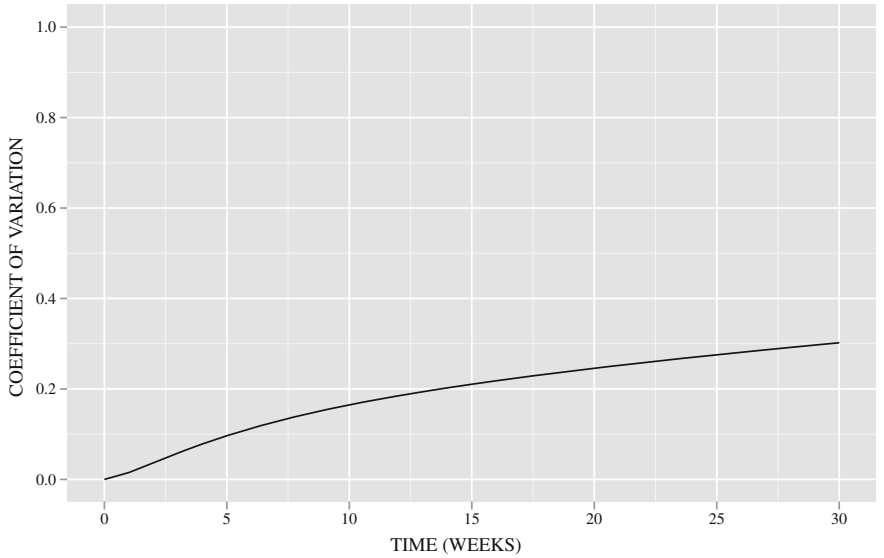

Fig. 11.41 The estimated coefficient of variation, $CV(t)$, in case of $\lambda(t) = \lambda_1(t)$

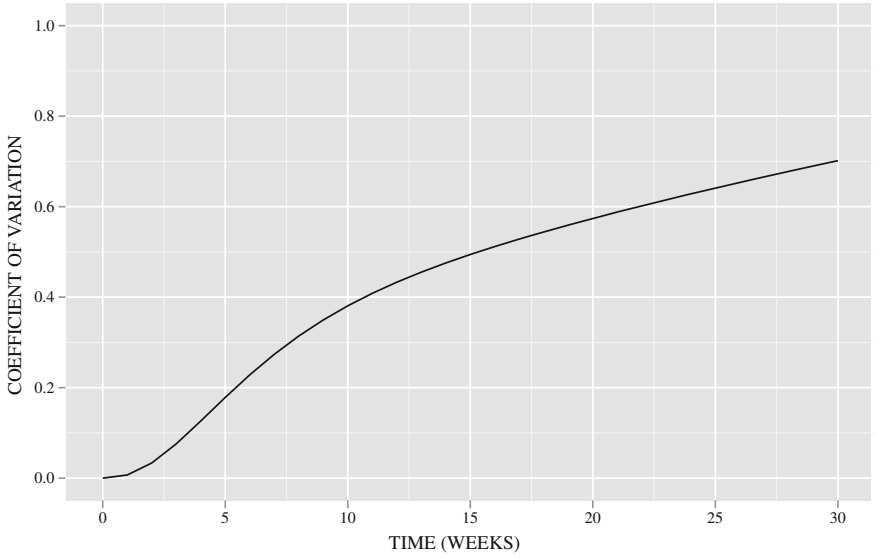

Fig. 11.42 The estimated coefficient of variation, $CV(t)$, in case of $\lambda(t) = \lambda_2(t)$

11.5.3 Discussion for the Method of Reliability Assessment on the Open Source Solution

We have focused on the open source solution which is known as the large-scale software system, and discussed the method of reliability assessment for the open source solution developed under several OSS's.

Moreover, we have proposed a software reliability growth model based on stochastic differential equations in order to consider the active state of the open source project and the collision among the open source components. Especially, we have assumed that the software failure intensity depends on the time, and the software fault-reporting phenomena on the bug tracking system keep an irregular state. Also, we have analyzed actual software fault-count data to show numerical examples of software reliability assessment for the large-scale open source solution. Moreover, we have derived several reliability assessment measures from our model.

At present, a new paradigm of distributed development typified by such open source project will evolve at a rapid pace in the future. Especially, it is difficult for the software testing managers to assess the reliability for the large-scale open source solution as a typical case of next-generation distributed development paradigm. Our method may be useful as the method of reliability assessment for the large-scale open source solution.

References

1. Olivier Fourdan, Xfce—Desktop Environment, http://www.xfce.org/
2. Fedora Project, sponsored by Red Hat. Online, Available: http://fedora.redhat.com/
3. The Apache HTTP Server Project, The Apache Software Foundation. Online, Available: http://httpd.apache.org/
4. S. Yamada, M. Kimura, H. Tanaka, S. Osaki, Software reliability measurement and assessment with stochastic differential equations. IEICE Trans. Fundam. **E77-A**(1), 109–116 (1994)
5. Open Handset Alliance, Android. Online, Available: http://www.android.com/
6. E. Andersen, BUSYBOX. Online Available: http://www.busybox.net/
7. G.J. Schick, R.W. Wolverton, An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **SE-4**(2), 104–120 (1978)
8. Z. Jelinski, P.B. Moranda, Software reliability research, in statistical computer performance evaluation, *Freiberger* (Academic Press, New York, 1972), pp. 465–484
9. P.B. Moranda, Event–altered rate models for general reliability analysis. IEEE Trans. Reliab. **R-28**(5), 376–381 (1979)
10. M. Xie, On a Generalization of the J-M model. Proc. Reliab. '89 **5**, Ba/3/1–5 Ba/3/7 (1989)
11. S. Yamada, S. Osaki, Cost-reliability optimal software release policies for software systems. IEEE Trans. Reliab. **R-34**(5), 422–424 (1985)
12. S. Yamada, S. Osaki, Optimal software release policies with simultaneous cost and reliability requirements. Eur. J. Oper. Res. **31**(1), 46–51 (1987)
13. Apache Tomcat, The Apache Software Foundation. Online, Available: http://tomcat.apache.org/
14. MySQL, Oracle Corporation and/or its Affiliates. Online, Available: http://www.mysql.com/

Chapter 12
Performance Illustrations of Software Tool

12.1 Embedded OSS

12.1.1 Applied Data

In this section, we analyze a set of actual software failure-occurrence time-interval data to show performance illustrations of software reliability/portability measurement for application of our tool.[1] We show numerical examples for reliability/portability assessment of the porting-phase of embedded system development by using embedded OSS. In this section, we analyze actual software failure time data to show numerical examples of software reliability measurement and assessment for application of our tool.

Especially, we focus on the embedded OSS in order to evaluate the performance of our tool, i.e., Android [2] and BusyBox [3]. BusyBox includes 4 components. Then, we show numerical examples of software reliability analysis for the Android OS developed for mobile phone. Also, we consider the case of installing BusyBox to Android as the porting environment. We use the data collected from the version in terms of "Android 1.5 NDK, Release 1" and " BusyBox 1.10.1 (stable)" as shown in Figs. 11.24 and 11.25. Thus, we illustrate the estimation results of our tool for the porting-phase assumed the above-mentioned porting environment.

The estimated result of the unknown parameters of our hazard rate model and the calculated result of MSE are shown in Fig. 12.1. From Fig. 12.1, our model fits better than the conventional hazard rate models in terms of the value of MSE. Also, Fig. 12.2 shows the porting stability. From Fig. 12.2, we find that the degree of maturity of Android OS means the low level from the estimated model parameters. Here, the degree of maturity of OSS means the low level, if w_1 is large. On the other hand, it means the high level, if w_1 is small.

[1] Reliability/Portability Assessment Tool for Embedded OSS (RPAT for Embedded OSS), URL: http://sourceforge.net/projects/rpatforeoss/ [1].

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_12

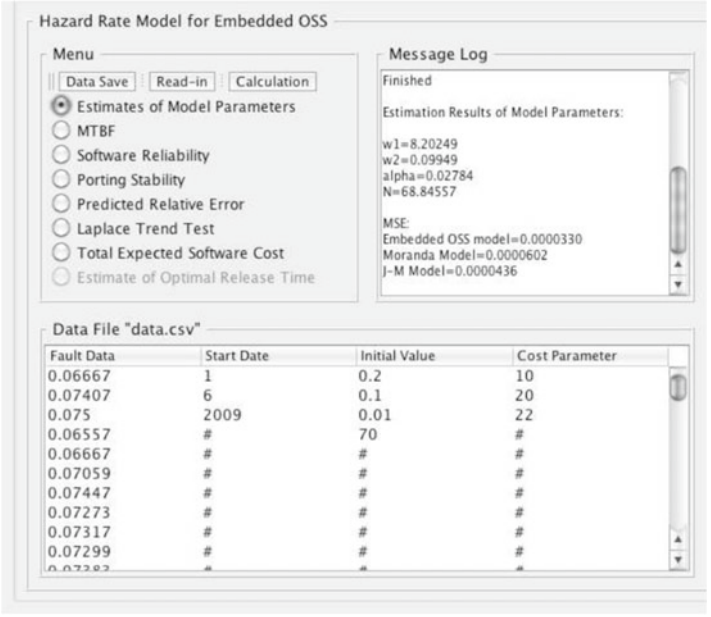

Fig. 12.1 The estimated results of the unknown parameters of our model

Also, the estimated MTBF's for our model in Eqs. (8.1)–(8.3), Moranda model, and J-M model are shown in Fig. 12.3. From Fig. 12.3, we can confirm that the MTBF grows as porting procedures go on. Moreover, Fig. 12.4 shows the estimated software reliability in Eq. (8.7). From Fig. 12.4, if the embedded system is operated after the 28 software failures, the software reliability at 2 days after from the beginning of its operation is calculated as about 0.15. Therefore, we find that at least one software failure may occur with the probability of 50 % within 0.5 day. Next, Fig. 12.5 shows the behavior of the predicted relative error in Eq. (8.17). As shown in Fig. 12.5, the variation of the modeling estimation becomes stable when the porting progress ratio exceeds 75 %.

Moreover, the estimation result the Laplace trend test statistics in Eq. (8.18) is shown in Fig. 12.6. From Fig. 12.6, the Laplace trend test shows a reliability regression trend in all area before failure No. 5. On the other hand, the Laplace trend test shows a reliability growth trend in all area after failure No. 5. We find that Fig. 12.6 includes both the reliability regression and growth trend. This implies that it is difficult to apply the conventional hazard rate models.

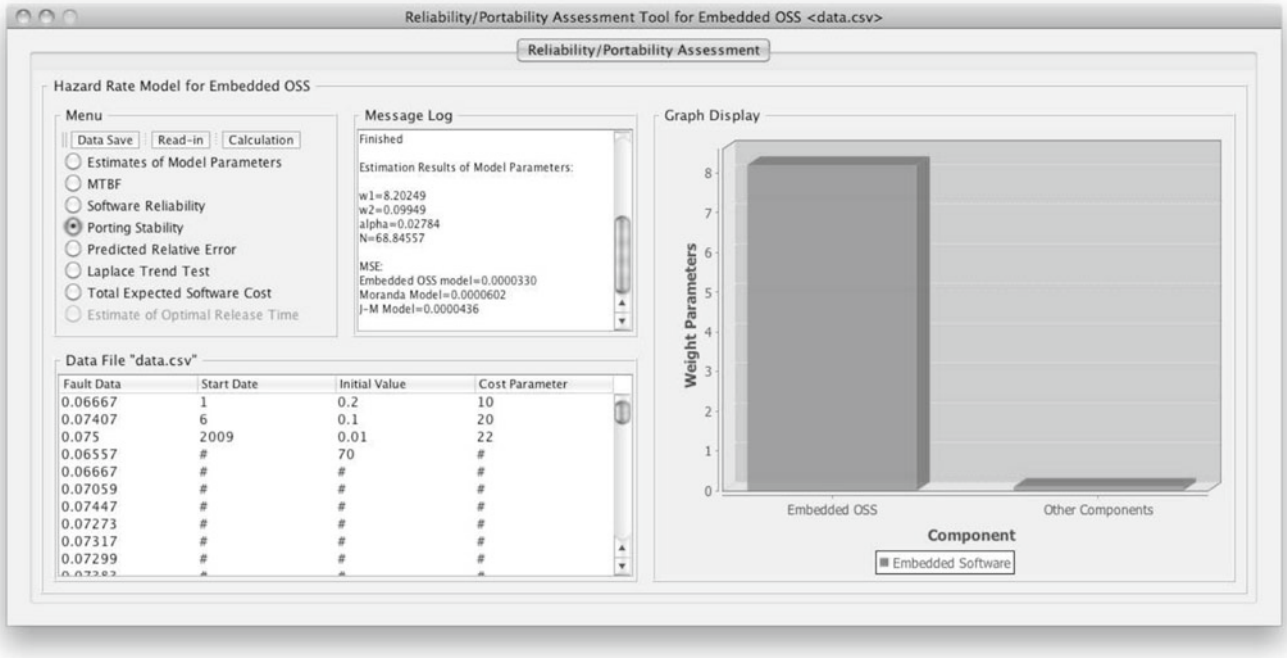

Fig. 12.2 The estimated porting stability

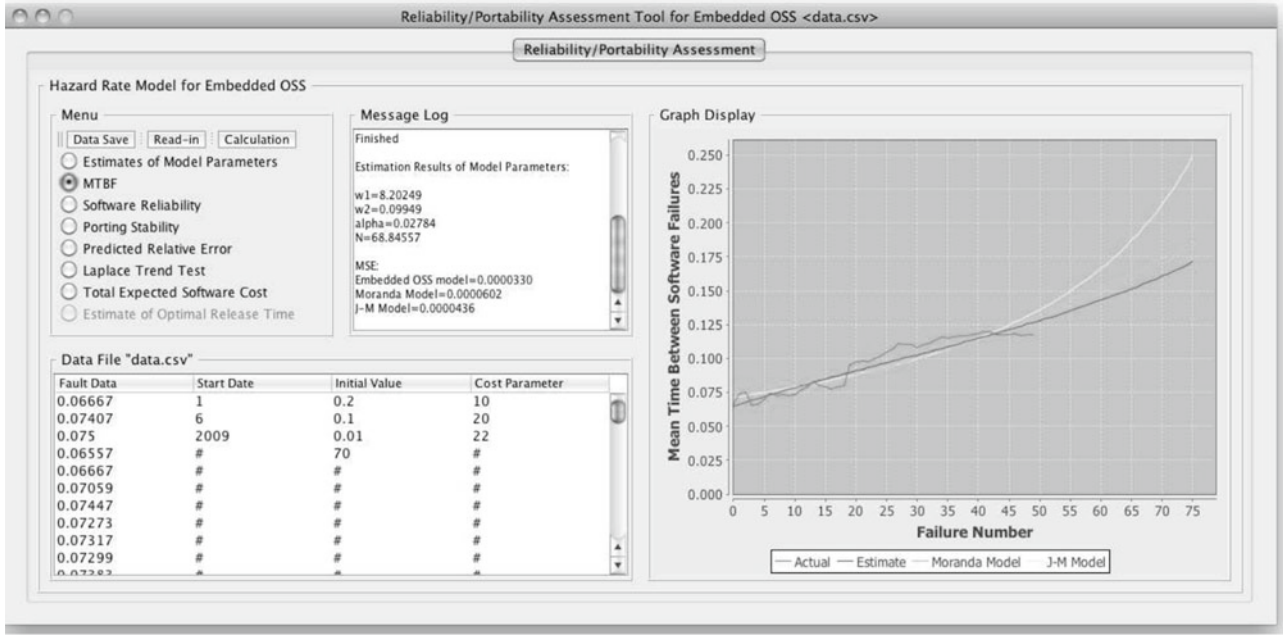

Fig. 12.3 The estimated MTBF

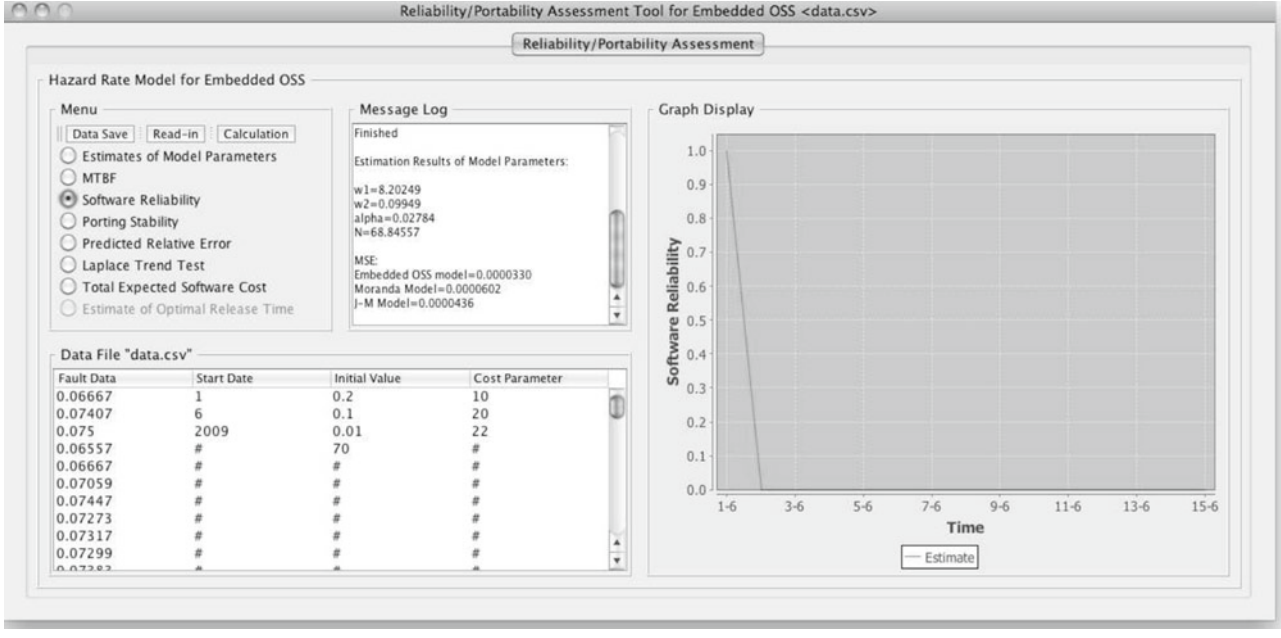

Fig. 12.4 The estimated software reliability

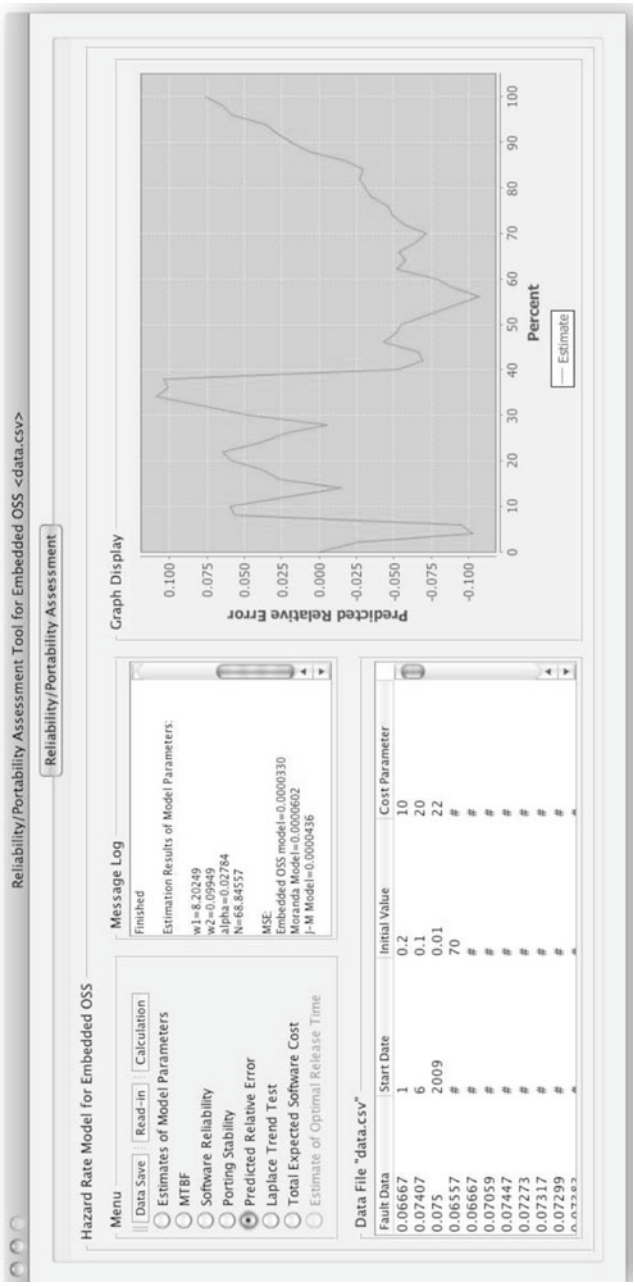

Fig. 12.5 The results of the predicted relative error

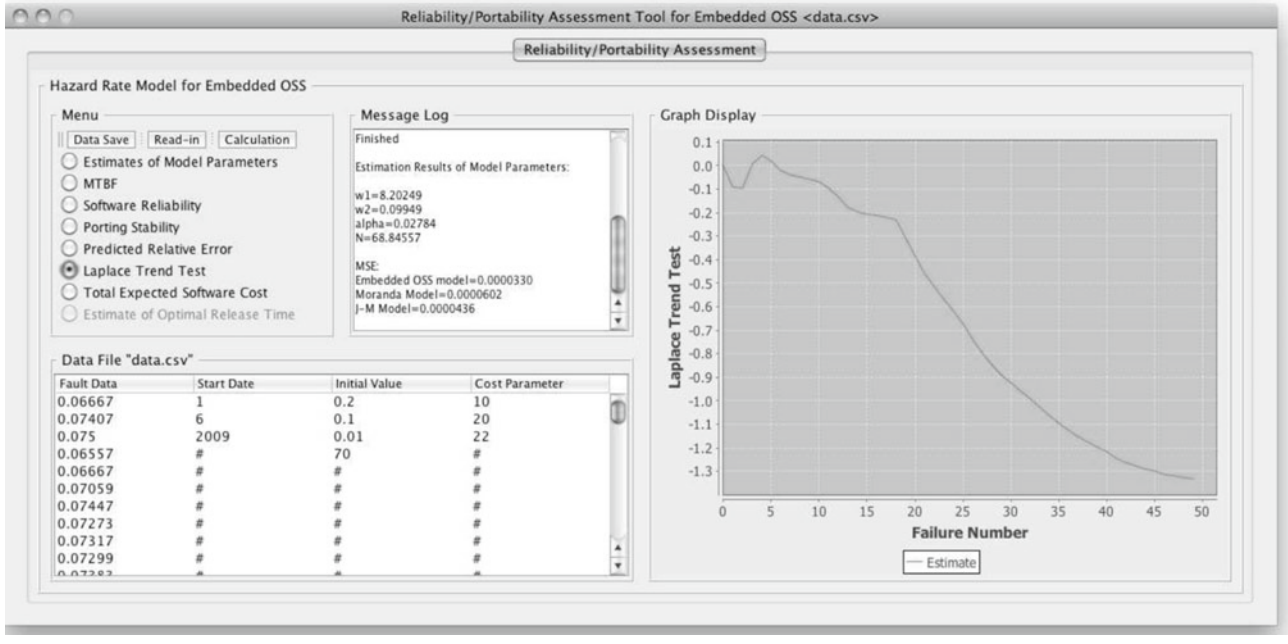

Fig. 12.6 The results of the Laplace trend test

12.1.2 Optimal Release Problem with Reliability of Embedded OSS

Also, we consider that the developer of embedded system can estimate the optimal release time by combining the total expected software cost with the reliability. We assume that the reliability requirement as follows:

$$R_k(0.1) = 0.5.$$

We obtain the reliability $R_{51}(0.1) = 0.462$ when the optimum release time is $t' = 6.6$ days where $l' = 51$. Then, we need to lengthen the porting-time, which is shown in Fig. 12.8 as $(t' - t^*) = 8.8 - 6.6 = 2.2$ days. Figure 12.8 illustrates the optimum release time with reliability objective $R_{61}(0.1) = 0.5$.

From Figs. 12.7 and 12.8, we have found that the optimum release time become lengthen, and the total expected software cost increases. When the porting time and the reliability objective are assumed as $R_k(0.1) = 0.5$, we obtain $t'^* = 8.8$ from $l^* = 61$. Then, the total expected software cost is 1502.8. Above-mentioned results are expressed as the tooltip of GUI in our tool.

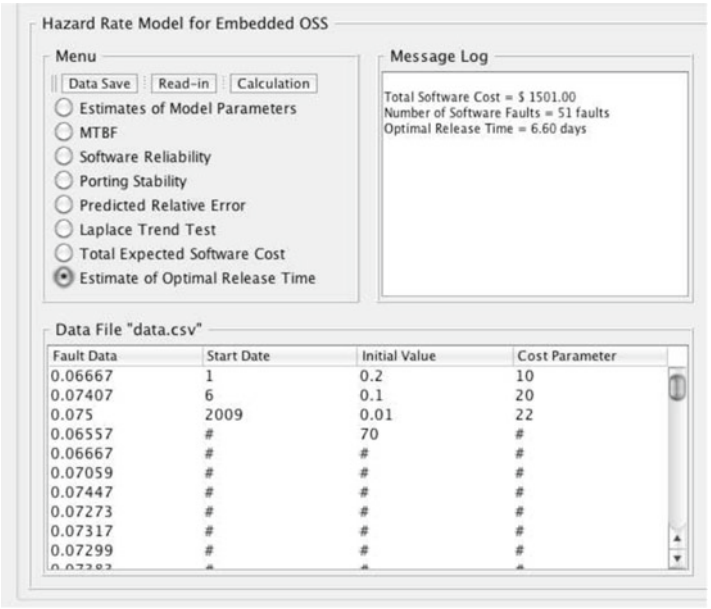

Fig. 12.7 The estimated results of the optimum software release time

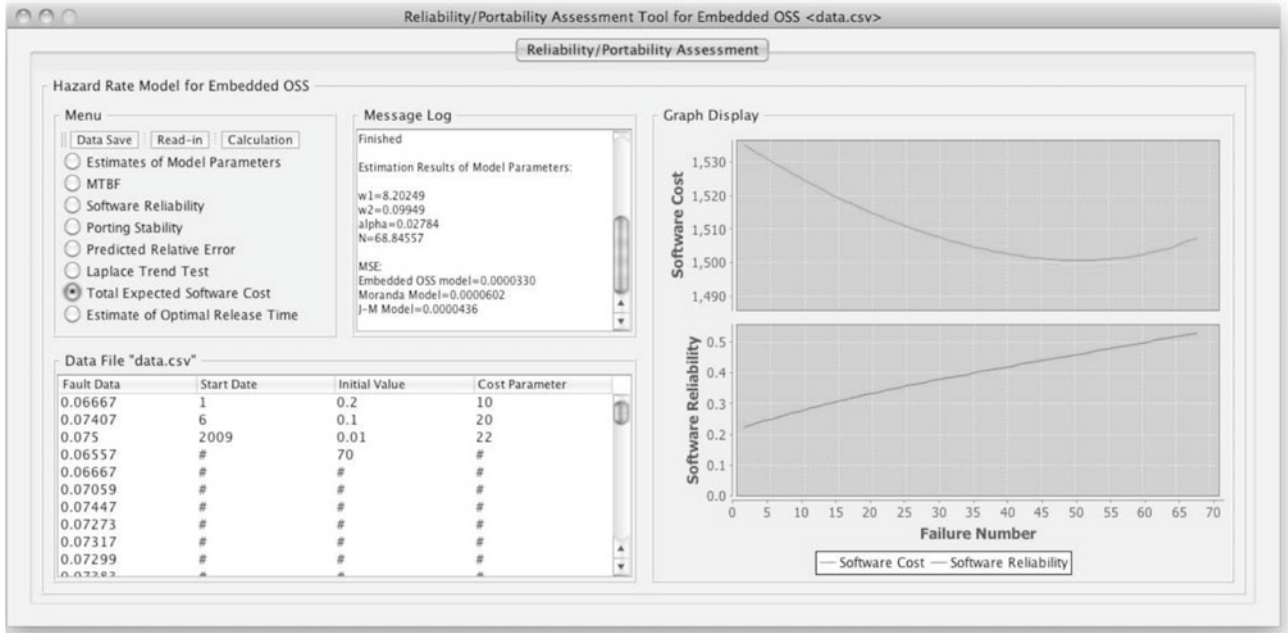

Fig. 12.8 The optimum software release time with reliability requirement, t'^{*}

12.1.3 Discussion for the Software Tool

It is important for software developers to control the porting-phase in embedded system development by using software reliability/portability assessment tool without knowing the details of the process of the software failure data analysis. In this section, we have developed a software reliability/portability assessment tool considering the optimal release problem based on the hazard rate model for embedded system development by using Java programming language.

We have shown that our method can grasp both the embedded OSS and the unique software components such as the device driver. Additionally, we have presented several performance illustrations for the actual data. Moreover, it is useful for embedded software developers to understand the debugging progress in porting-phase of embedded system development by using a software reliability/portability assessment tool without knowing the details of the process of the software failure data analysis. Furthermore, our tool proposed here is useful for embedded system developers in terms of the management of the debugging process in the porting-phase of embedded system development. The results of data analysis are represented simply and visually by GUI and this tool prepares expandability, portability, and maintainability by using Java.

Finally, we have focused on the reliability/portability under the porting-phase of an embedded system development by using the embedded OSS. Distributed development environment typified by such embedded OSS will evolve at a rapid pace in the future. Our reliability/portability assessment tool may be useful as a method of reliability/portability assessment to solve the problems that many companies have been hesitant to innovate the embedded OSS.

12.2 Open Source Solution

12.2.1 Applied Data

A set of actual software fault-detection count data is analyzed to show performance illustrations of software reliability measurement for application of the developed tool.[2] This tool [4] is developed by using open source Flex SDK. The open source Flex is a powerful application framework that allows developers to easily build mobile applications for iOS, Android, and BlackBerry Tablet OS, as well as traditional applications for the browser and the desktop by using the same programming model, the same tool, and the same code base [5].

[2]Reliability Analysis Tool Based on Stochastic Differential Equation Models for Open Source Solution (RAT Based on SDE Models for OSSol), URL: http://sourceforge.net/projects/ossol/.

Several numerical examples for reliability analysis of the debugging-process of open source solution development are shown in this section. Moreover, the estimation results of the developed tool are illustrated by using the data sets assumed the debugging environment. Considering the realities of the field use, we show the numerical examples by using the data sets in terms of the Apache HTTP server [6], Apache Tomcat [7], and MySQL [8].

The estimated result of the unknown parameters of the proposed SDE models is shown in Fig. 12.9. Figure 12.9 shows that the proposed SDE models fit better than the conventional SDE models in terms of AIC and MSE. Also, the estimated expected number of detected faults in Eqs. (9.11) and (9.12), and the conventional ones are shown in Fig. 12.10. From Fig. 12.10, the S-shaped growth curves rapidly converge as debugging procedures go on. On the other hand, the exponential growth curves slowly converge.

Figures 12.11 and 12.12 show the estimated sample path of detected faults. In particular, Fig. 12.11 means that the complexity of component collision is large in the early debugging-process of open source solution. On the other hand, Fig. 12.12 shows that the noise becomes large as debugging procedures go on. Above mentioned results, Fig. 12.11 has the characteristic of the binding-phase of open source solution.

Also, Fig. 12.13 shows the behavior of the predicted relative error in Eq. (8.17). As shown in Fig. 12.13, the variation of all modeling estimations becomes stable when the debugging progress ratio exceeds 50 %.

Furthermore, it is important for software managers to assess the expected number of remaining faults. The expected number of remaining faults for each model are shown in Fig. 12.14. Especially, the estimated expected number of remaining faults in terms of S-shaped growth curve are estimated optimistically.

12.2.2 Discussion for the Software Tool

This section focuses on the open source solution which is known as the large scale software system, and discusses the method of reliability assessment for the open source solution developed under several OSS's.

The method of software reliability analysis based on SDE models has been proposed in order to consider the collision among the OSS components. Then, the proposed models have assumed that the software fault-detection rate depends on the time, and the software fault-report phenomena on the debugging-process keep an irregular state. Especially, the reliability analysis tool based on SDE models in order to consider the interesting aspect of the collision status in the binding phase of OSS's has developed. Also, a set of actual software fault-count data has been analyzed to show numerical examples of software reliability analysis for the open source solution.

At present, a new paradigm of distributed development typified by such open source project will evolve at a rapid pace in the future. Especially, it is difficult for the software managers to assess the reliability for the large scale open source solution

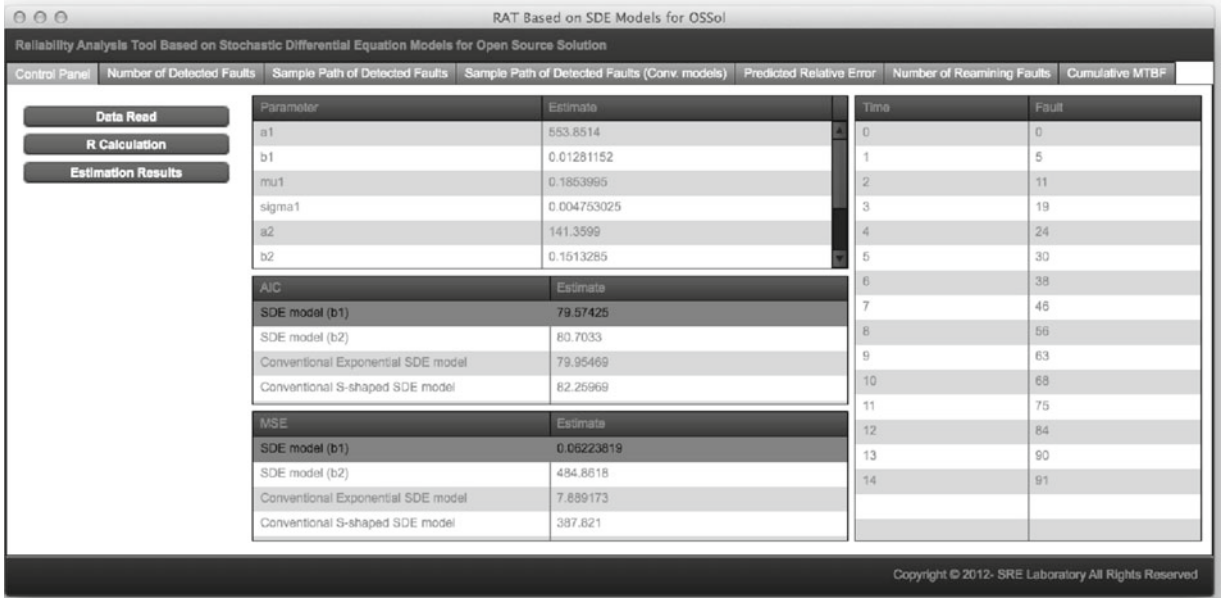

Fig. 12.9 The estimation results of model parameters, AIC, and MSE

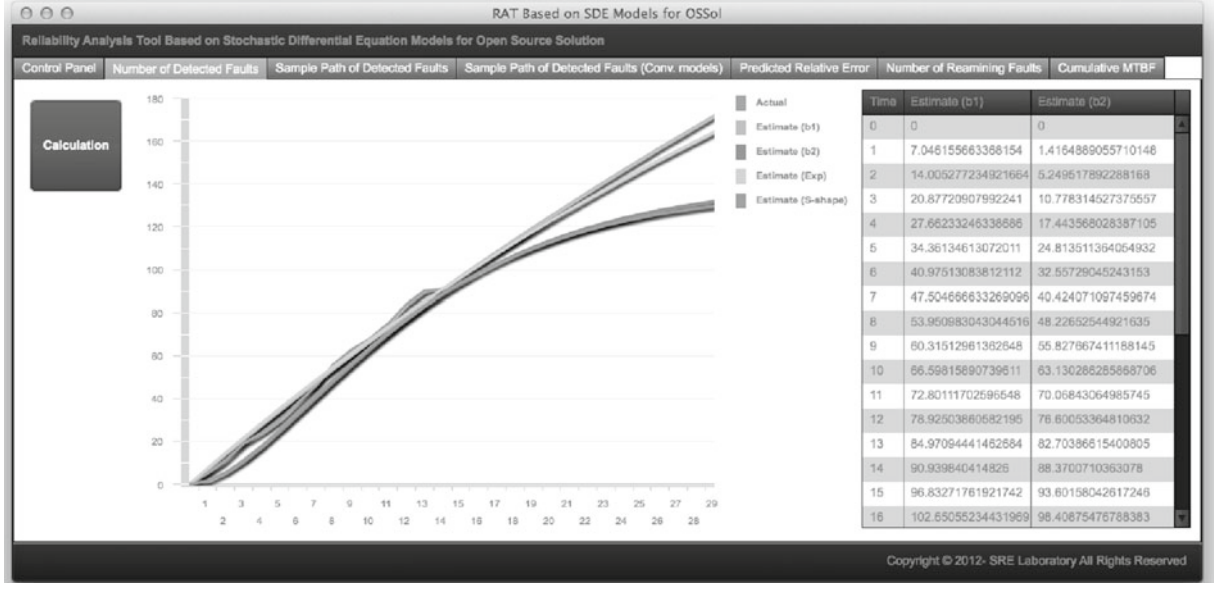

Fig. 12.10 The estimated expected number of detected faults

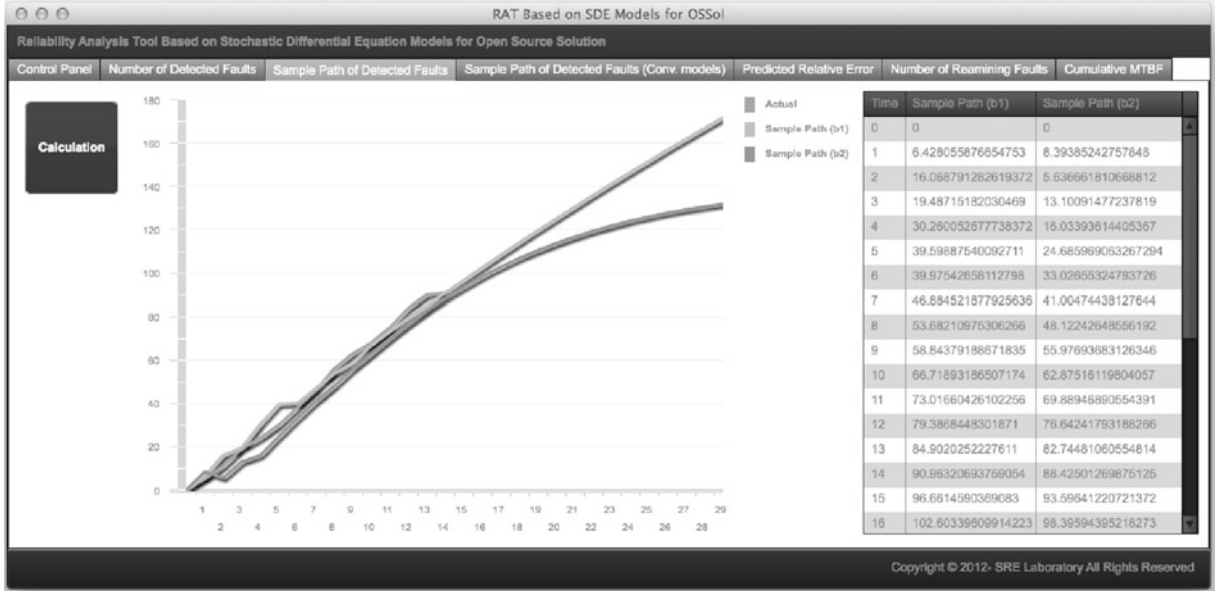

Fig. 12.11 The sample path of cumulative number of detected faults in the proposed SDE models

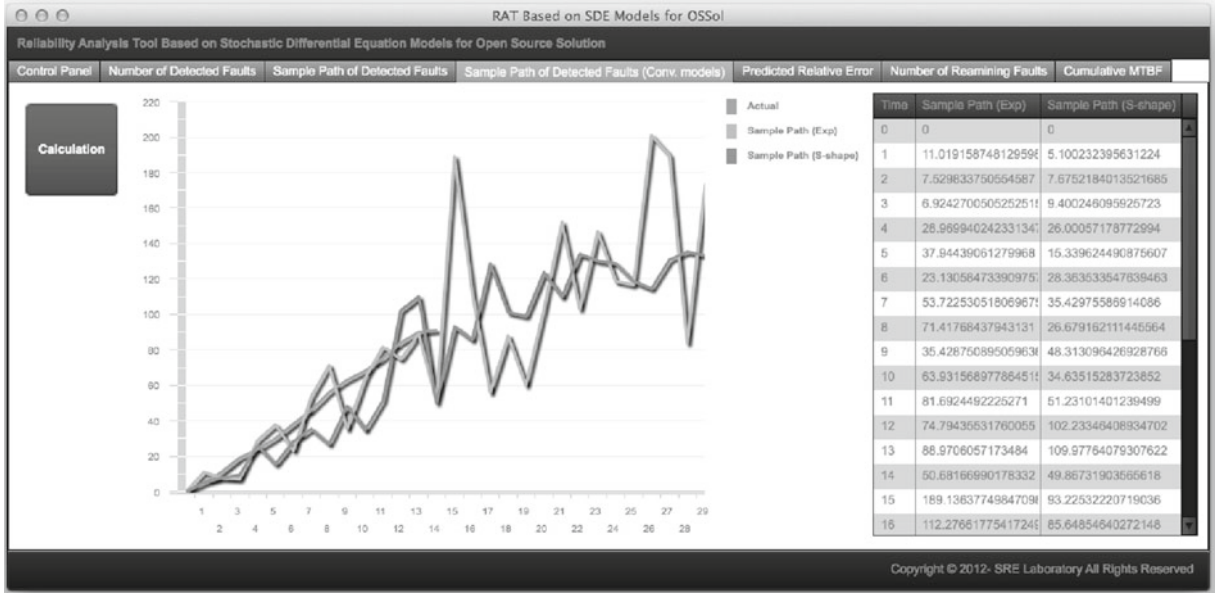

Fig. 12.12 The sample path of cumulative number of detected faults in the conventional SDE models

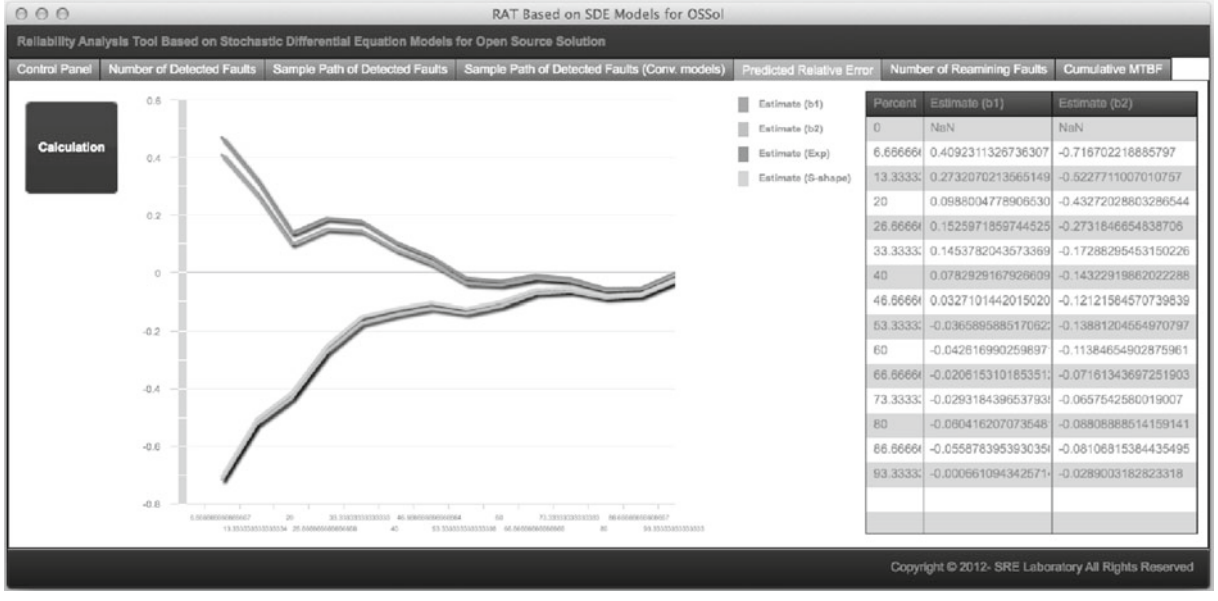

Fig. 12.13 The estimated predicted relative error

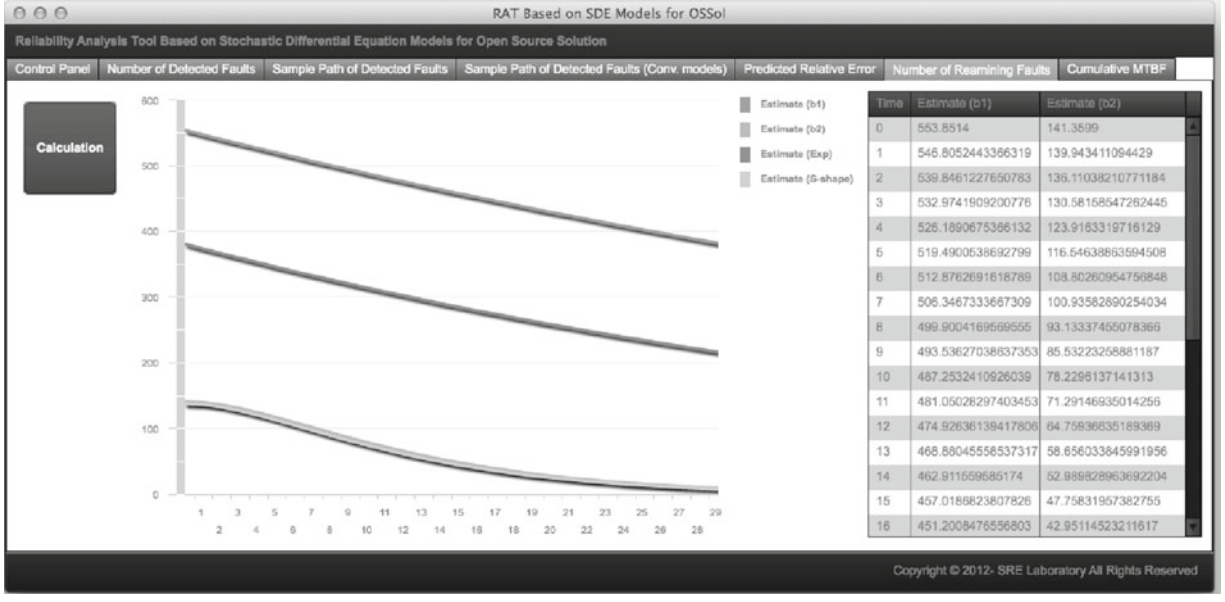

Fig. 12.14 The estimated expected number of remaining faults

as a typical case of next-generation distributed development paradigms. The proposed method may be useful as the method of software reliability assessment for the open source solution.

12.3 Mobile OSS

12.3.1 Applied Data

A set of actual software fault-detection count data is analyzed to show performance illustrations of software reliability analysis for application of the developed tool. This tool [9] is developed by using open source Apache Flex SDK. The open source Apache Flex is a powerful application framework that allows developers to easily build mobile applications for iOS, Android, and BlackBerry Tablet OS, as well as traditional applications for the browser and the desktop using the same programming model, the same tool, and the same code base [5]. Several numerical examples for reliability analysis of the operation-phase of mobile clouds environment are shown in this section. In particular, we focus on Firefox OS [10] in order to evaluate the performance of our method and tool. In this section, we show numerical examples by using the data sets for Firefox OS of OSS mobile software. This data are collected in the bug tracking system on the website of Mozilla as Firefox OS project. Table 12.1 shows actual data sets.

Also, we focus on the network traffic as the characteristics of mobile clouds. Then, we consider that the network traffic changes as depending on the environment for the usage of mobile clouds. In particular, we assume the following random number for the change as the network traffic density in order to aim at assessing the performance of proposed method, because it is difficult to obtain actual network traffic data in mobile clouds used several mobile software simultaneously.

- Normal Random Number
  (Mean :0.0, Standard Deviation: 0.5)

Then, we focus on two cases of version 1.1 and 1.2 in Firefox OS.

We show the main screen in case of version 1.1 of the developed dynamic reliability analysis tool in Fig. 12.15. For examples of dynamic reliability analysis, it is important for software managers to understand the trends of reliability growth or regression in terms of the number of detected faults according to the change in the network traffic. Then, Fig. 12.16 show the estimated network traffic density. Also, the estimated MTBF in Eq. (10.4) is shown in Fig. 12.17. From Figs. 12.16 and 12.17, we find that the mobile software of version 1.1 is under the influence of network environment in the early debugging-process of mobile clouds. Moreover, the mobile clouds in case of version 1.1 tends to show the trend of reliability regression because the estimated MTBF is nearly unchanged. Furthermore, Fig. 12.18 shows the estimated software reliability in Eq. (10.3) in case of version 1.1. The horizontal axis

Table 12.1 The actual MTBF data in Firefox OS

Failure number	MTBF (Days)	Failure number	MTBF (Days)
1	40.2590162	40	2.222743056
2	0.065416667	41	2.771863426
3	0.003900463	42	3.033171296
4	0.031006944	43	6.515335648
5	0.000729167	44	0.008425926
6	1.558958333	45	0.003252315
7	22.02296296	46	0.001238426
8	13.54936343	47	0.003576389
9	0.442013889	48	0.000844907
10	5.431226852	49	0.000763889
11	20.11006944	50	0.000671296
12	6.572094907	51	0.000740741
13	14.09141204	52	0.000856481
14	3.334606481	53	0.002071759
15	5.527094907	54	0.006365741
16	0.629259259	55	0.004664352
17	6.252430556	56	0.006990741
18	0.764722222	57	0.327291667
19	2.643587963	58	1.717696759
20	0.884675926	59	5.71556713
21	0.295405093	60	0.059594907
22	0.621388889	61	0.859201389
23	0.779710648	62	6.872789352
24	1.025358796	63	0.016736111
25	0.794548611	64	0.004791667
26	1.101215278	65	1.35806713
27	2.453101852	66	5.848252315
28	0.330613426	67	1.783587963
29	0.50630787	68	0.790173611
30	0.321828704	69	2.495358796
31	0.090509259	70	0.005949074
32	0.012881944	71	4.276087963
33	0.072314815	72	5.475706019
34	0.749305556	73	0.822719907
35	0.061701389	74	0.255115741
36	0.122743056	75	5.923206019
37	1.312372685	76	3.86837963
38	0.368564815	77	0.192141204
39	2.932361111	78	0.048391204

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
79	0.886458333	118	0.011875
80	0.872476852	119	0.03525463
81	0.664340278	120	0.083229167
82	0.020300926	121	0.004351852
83	0.343263889	122	0.00412037
84	0.093402778	123	0.003854167
85	0.148865741	124	0.128217593
86	0.698831019	125	0.00474537
87	0.251238426	126	0.272314815
88	0.027615741	127	2.003101852
89	0.017662037	128	0.640266204
90	0.070219907	129	0.988958333
91	0.024212963	130	0.062928241
92	0.831956019	131	0.024849537
93	0.194965278	132	1.10099537
94	0.528599537	133	0.09412037
95	1.016736111	134	0.117037037
96	0.021226852	135	0.029861111
97	0.049247685	136	0.706412037
98	0.004340278	137	0.146539352
99	0.19962963	138	0.834583333
100	0.010462963	139	1.127523148
101	0.062962963	140	0.393472222
102	0.010671296	141	0.036909722
103	0.133946759	142	2.277013889
104	0.012974537	143	0.508668981
105	0.007731481	144	0.196956019
106	0.449328704	145	0.055509259
107	0.075185185	146	0.01525463
108	0.363611111	147	0.967581019
109	0.208483796	148	0.386770833
110	0.293888889	149	0.135833333
111	0.277800926	150	0.198564815
112	0.1628125	151	1.073553241
113	0.617731481	152	0.526261574
114	0.008333333	153	0.036539352
115	0.012303241	154	0.413993056
116	0.011064815	155	0.579155093
117	0.059282407	156	2.435104167

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
157	0.393344907	196	0.004456019
158	0.098009259	197	0.007430556
159	0.006898148	198	0.005972222
160	0.004050926	199	0.081365741
161	0.670520833	200	0.003263889
162	1.040752315	201	0.034166667
163	0.057002315	202	0.200775463
164	0.815023148	203	0.125729167
165	0.086377315	204	0.011099537
166	0.060034722	205	0.027233796
167	0.009027778	206	0.449224537
168	0.085949074	207	0.046689815
169	0.449143519	208	0.035891204
170	2.597303241	209	0.394143519
171	0.185868056	210	0.029664352
172	0.072604167	211	0.093668981
173	0.059375	212	0.366319444
174	0.582395833	213	2.238564815
175	0.014074074	214	0.047384259
176	0.199907407	215	0.006099537
177	0.123773148	216	0.007094907
178	0.011921296	217	0.000659722
179	0.273136574	218	0.031481481
180	0.463078704	219	0.021319444
181	0.19525463	220	0.038206019
182	0.024513889	221	0.01224537
183	0.007939815	222	0.049780093
184	0.001331019	223	0.008113426
185	0.004571759	224	0.00431713
186	0.009618056	225	0.066145833
187	0.048402778	226	0.016863426
188	0.034212963	227	0.583449074
189	0.019965278	228	0.242453704
190	0.078391204	229	0.027638889
191	2.31E-05	230	0.022083333
192	0.302222222	231	0.013912037
193	0.093518519	232	0.03318287
194	0.003449074	233	0.031840278
195	0.065729167	234	0.10599537

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
235	0.019421296	274	0.193599537
236	0.697696759	275	1.106076389
237	0.127962963	276	0.139965278
238	0.462337963	277	0.032997685
239	0.20931713	278	1.144363426
240	0.017939815	279	0.224722222
241	0.146076389	280	1.448414352
242	0.017986111	281	0.870532407
243	0.045150463	282	0.053819444
244	0.024016204	283	0.653171296
245	0.02349537	284	0.258958333
246	0.118240741	285	0.05099537
247	0.097222222	286	0.08337963
248	0.146747685	287	0.005162037
249	0.601284722	288	0.167083333
250	0.736863426	289	0.120763889
251	0.054560185	290	0.002569444
252	0.211458333	291	0.010162037
253	0.326064815	292	0.003576389
254	0.014375	293	0.002569444
255	1.595127315	294	0.004282407
256	0.00625	295	0.034953704
257	0.011296296	296	0.104814815
258	0.096967593	297	0.16099537
259	0.083564815	298	0.084710648
260	0.017534722	299	0.159722222
261	0.039212963	300	0.36462963
262	0.000810185	301	0.116296296
263	0.321122685	302	0.009259259
264	0.234895833	303	0.089791667
265	0.047025463	304	0.076388889
266	0.068287037	305	0.029131944
267	0.027013889	306	0.006585648
268	0.031516204	307	0.391770833
269	0.111712963	308	0.066863426
270	0.330474537	309	0.242060185
271	0.328229167	310	0.030752315
272	0.201053241	310	0.030752315
273	0.036145833	311	0.065717593

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
312	0.178298611	351	0.09212963
313	0.4440625	352	0.055300926
314	0.132650463	353	0.021215278
315	0.190034722	354	0.070763889
316	0.047939815	355	0.331875
317	0.135127315	356	0.20037037
318	0.56724537	357	0.070613426
319	0.006678241	358	0.292858796
320	0.002638889	359	1.97625
321	0.004618056	360	0.106550926
322	1.700821759	361	0.137777778
323	0.2578125	362	0.003923611
324	0.039212963	363	0.318935185
325	0.310451389	364	0.158715278
326	0.102835648	365	0.033391204
327	0.548541667	366	0.013414352
328	0.178148148	367	0.040891204
329	0.18994213	368	0.013113426
330	0.08724537	369	0.339456019
331	0.101145833	370	0.045358796
332	0.354467593	371	0.48625
333	0.018009259	372	0.080925926
334	0.022789352	373	0.262453704
335	0.294733796	374	0.008622685
336	0.024699074	375	0.022303241
337	0.864699074	376	0.034606481
338	0.013831019	377	0.072233796
339	0.105150463	378	0.037222222
340	0.005231481	379	0.00625
341	0.002453704	380	0.026400463
342	0.006180556	381	0.451527778
343	0.005150463	382	0.393854167
344	0.001527778	383	0.007002315
345	0.001550926	384	0.17787037
346	0.002534722	385	0.007349537
347	0.059641204	386	0.087766204
348	0.284780093	387	0.008217593
349	0.005023148	388	0.180844907
350	0.047141204	389	0.068831019

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
390	0.169212963	429	0.135474537
391	0.051030093	430	0.004814815
392	0.053611111	431	0.053958333
393	0.002476852	432	0.157800926
394	0.014155093	433	0.100358796
395	0.005196759	434	0.29412037
396	0.004409722	435	0.097777778
397	0.014398148	436	0.104895833
398	0.0075	437	0.039965278
399	0.000150463	438	0.017881944
400	0.002592593	439	0.063055556
401	0.004537037	440	0.016770833
402	0.076064815	441	0.117361111
403	0.003020833	442	0.213263889
404	0.128483796	443	0.117546296
405	0.113020833	444	0.466643519
406	0.242662037	445	0.149907407
407	0.555208333	446	0.175601852
408	1.066539352	447	1.373831019
409	0.794861111	448	0.270972222
410	0.036770833	449	0.044606481
411	0.062743056	450	0.210532407
412	0.006921296	451	1.724652778
413	0.015196759	452	0.131400463
414	0.033101852	453	0.011111111
415	0.020925926	454	0.074027778
416	0.023368056	455	0.051412037
417	0.024479167	456	0.677604167
418	0.009618056	457	0.09599537
419	0.091643519	458	0.004166667
420	0.000324074	459	0.070636574
421	0.007928241	460	0.096087963
422	0.014340278	461	0.015104167
423	0.203657407	462	0.280173611
424	0.175069444	463	0.4975
425	0.162662037	464	0.050393519
426	0.084467593	465	0.05287037
427	0.035648148	466	0.007997685
428	0.073194444	467	0.002511574

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
468	0.00630787	507	0.196296296
469	0.069305556	508	0.641030093
470	0.270162037	509	0.032858796
471	0.236053241	510	0.013148148
472	0.377106481	511	0.022233796
473	0.04193287	512	0.008090278
474	0.038842593	513	0.020625
475	0.046076389	514	0.01244213
476	0.172615741	515	0.001701389
477	0.102152778	516	0.002905093
478	0.123703704	517	0.034456019
479	0.107337963	518	0.009953704
480	0.256342593	519	0.016782407
481	0.008078704	520	0.067662037
482	0.120729167	521	0.077013889
483	0.050231481	522	0.149618056
484	0.053622685	523	0.061365741
485	0.09775463	524	0.051458333
486	1.872766204	525	0.072465278
487	0.703819444	526	0.304918981
488	0.098553241	527	0.01125
489	0.206469907	528	0.066041667
490	0.038888889	529	0.001226852
491	0.044259259	530	0.00318287
492	0.138194444	531	0.014421296
493	0.395543981	532	0.022303241
494	0.007256944	533	0.016122685
495	0.018831019	534	0.069224537
496	0.057048611	535	0.057349537
497	0.018275463	536	0.006666667
498	0.044166667	537	0.233564815
499	0.018333333	538	0.210138889
500	0.068773148	539	0.192418981
501	0.038055556	540	0.114826389
502	0.002974537	541	0.01275463
503	0.013240741	542	0.004247685
504	0.022638889	543	0.085636574
505	0.022800926	544	0.001643519
506	0.02662037	545	0.156886574

(continued)

Table 12.1 (continued)

Failure number	MTBF (Days)	Failure number	MTBF (Days)
546	0.025856481	585	0.079837963
547	0.076666667	586	0.327361111
548	2.468645833	587	0.042696759
549	0.253900463	588	0.086053241
550	0.057581019	589	0.201342593
551	0.358148148	590	0.159780093
552	0.038043981	591	0.069236111
553	0.230335648	592	0.000763889
554	0.04724537	593	0.020752315
555	0.194421296	594	0.045868056
556	0.026550926	595	0.157037037
557	0.19462963	596	0.056365741
558	0.246527778	597	0.000416667
559	0.063402778	598	0.00625
560	0.193738426	599	0.072199074
561	0.045138889	600	0.073703704
562	0.312581019	601	0.548333333
563	0.029027778	602	0.422569444
564	0.093206019	603	0.00681713
565	0.002476852	604	0.564872685
566	0.098483796	605	0.061493056
567	0.0296875	606	0.909722222
568	0.095277778	607	0.322800926
569	0.082662037	608	0.146597222
570	0.016886574	609	0.231377315
571	0.052951389	610	0.495405093
572	0.073449074	611	0.234236111
573	0.057835648	612	0.722291667
574	0.311296296	613	1.050196759
575	0.120775463	614	0.12787037
576	0.157638889	615	0.045717593
577	0.047083333	616	0.023993056
578	0.342291667	617	0.860925926
579	0.30025463	618	0.098321759
580	0.198969907	619	0.025081019
581	0.249189815	620	0.582835648
582	0.318819444	621	0.067280093
583	0.021631944	622	0.266643519
584	2.132916667	623	0.56650463

(continued)

Table 12.1 (continued)

Failure Number	MTBF (Days)	Failure Number	MTBF (Days)
624	0.399548611	660	2.757511574
625	0.007337963	661	0.156435185
626	0.200393519	662	0.055844907
627	0.135648148	663	1.002106481
628	0.323020833	664	2.845694444
629	0.352233796	665	4.194606481
630	2.684571759	666	1.6359375
631	0.259803241	667	1.234837963
632	0.050787037	668	2.952777778
633	0.626203704	669	15.03015046
634	0.088784722	670	66.01752315
635	0.755520833		
636	0.128252315		
637	0.184710648		
638	1.635243056		
639	0.580752315		
640	0.005914352		
641	0.070601852		
642	0.100023148		
643	2.733310185		
644	1.448136574		
645	0.936076389		
646	0.780092593		
647	1.053865741		
648	2.729907407		
649	0.011423611		
650	0.008784722		
651	0.003900463		
652	0.022951389		
653	0.1159375		
654	0.104918981		
655	0.590972222		
656	1.156006944		
657	0.312199074		
658	0.253796296		
659	1.596875		

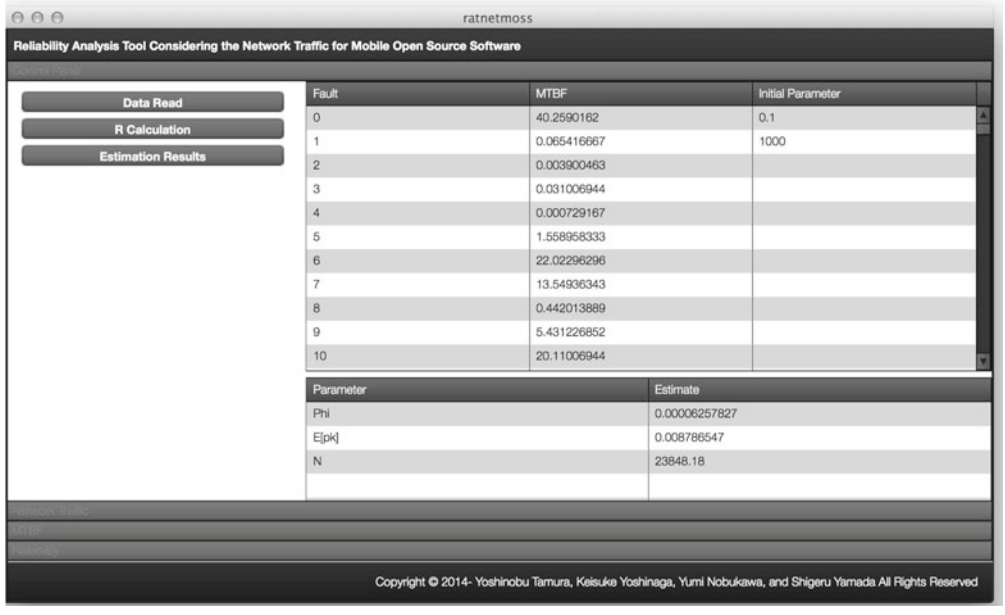

Fig. 12.15 The main screen of the developed AIR application in case of Firefox OS (version 1.1)

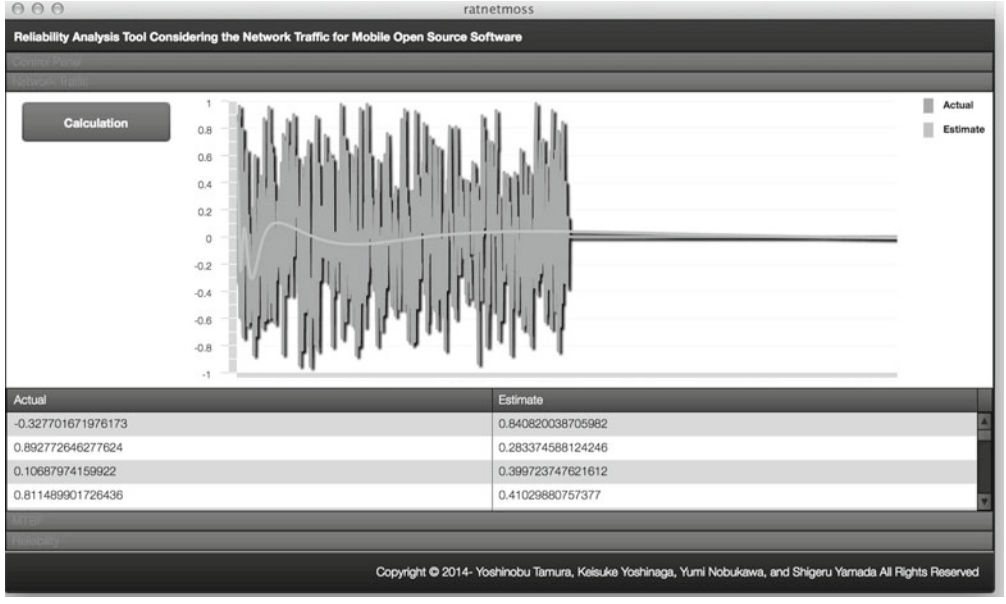

Fig. 12.16 The estimated network traffic density in case of Firefox OS (version 1.1)

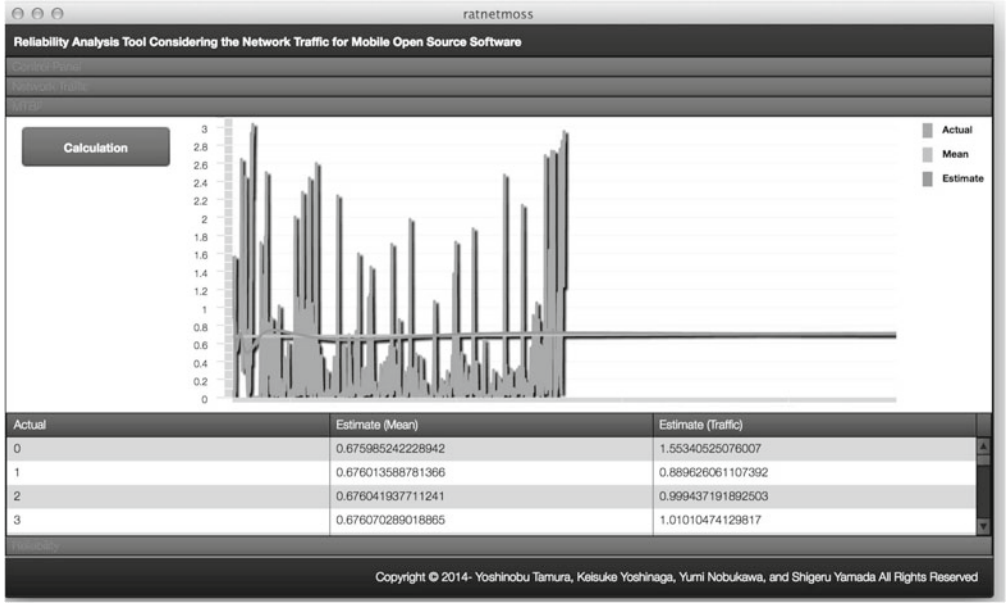

Fig. 12.17 The estimated MTBF in case of Firefox OS (version 1.1)

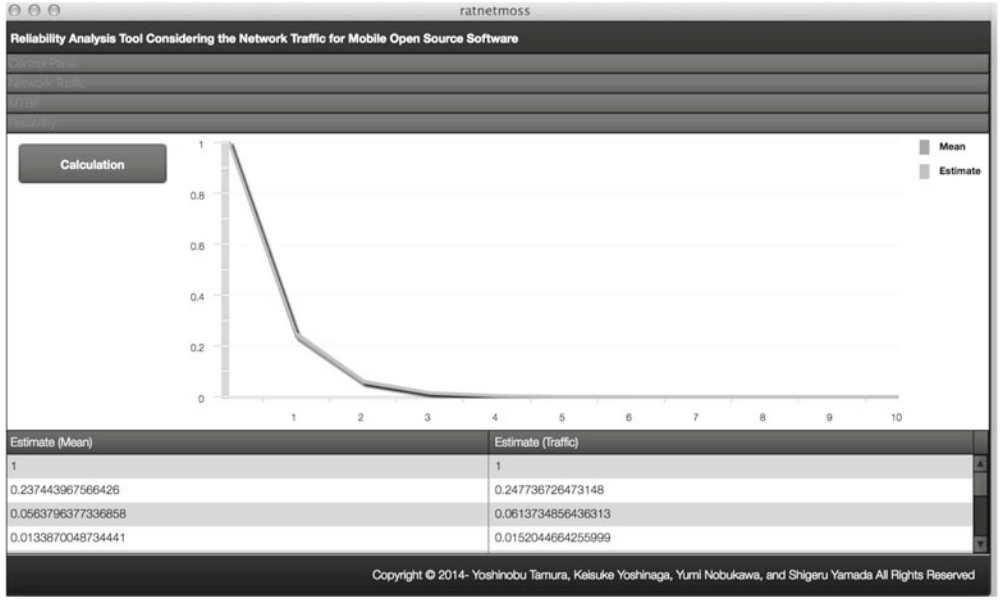

Fig. 12.18 The estimated software reliability in case of Firefox OS (version 1.1)

of Fig. 12.18 means the elapsed time after the end of the detected faults $k = 669$. From Fig. 12.18, if the embedded software is operated after the under the same strict conditions as the debugging-process after the 669 faults, the software reliability after 27 h after from the beginning of its operation shows about 0.2. Therefore, we find that at least one software failure may occur with the probability of 80 % within 27 h.

Similarly, the main screen in case of version 1.2 of Firefox OS is shown in Fig. 12.19. Figure 12.19 shows, the estimated parameters, the calculation button, etc. Also, Figs. 12.20, 12.21 and 12.22 show the estimated network traffic density, the estimated MTBF, the estimated software reliability, respectively. In particular, from Fig. 12.21, we find that the mobile software of version 1.2 is under the influence of network environment in the early debugging-process of mobile clouds. Moreover, the mobile clouds in case of version 1.2 tends to show the trend of reliability growth because the estimated MTBF becomes large with the operating procedures go on. Furthermore, Fig. 12.22 shows the estimated software reliability in Eq. (10.3) in case of version 1.2. The horizontal axis of Fig. 12.22 means the elapsed time after the end of the detected faults $k = 323$. From Fig. 12.22, if the embedded software is operated after the under the same strict conditions as the debugging-process after the 323 faults, the software reliability after 27 h after from the beginning of its operation shows about 0.4. Therefore, we find that at least one software failure may occur with the probability of 60 % within 27 h.

12.3.2 Discussion for the Software Tool

It is important for software developers and managers to control the debugging-process in mobile system development considering the mobile clouds. We have proposed the method of software reliability measurement and assessment considering the network traffic based on the hazard rate model and neural network for mobile clouds environment.

In particular, it is difficult to assess the reliability of the operation phase according to the installed software in the future, because the mobile device includes the software installer from the characteristics of third-party development paradigm. Then, we have proposed the method of reliability measurement and assessment for the mobile clouds. Moreover, we have developed the AIR application based on the proposed method. Additionally, we have presented several performance examples of the developed AIR application and the proposed method for the actual data. Moreover, it is important for software managers to assess the reliability according to the change of network traffic in the mobile device. We have shown the estimated MTBF and software reliability considering the change of network traffic. Thereby, we have found that the developed AIR application can assess integrated reliability considering both software failure and network traffic.

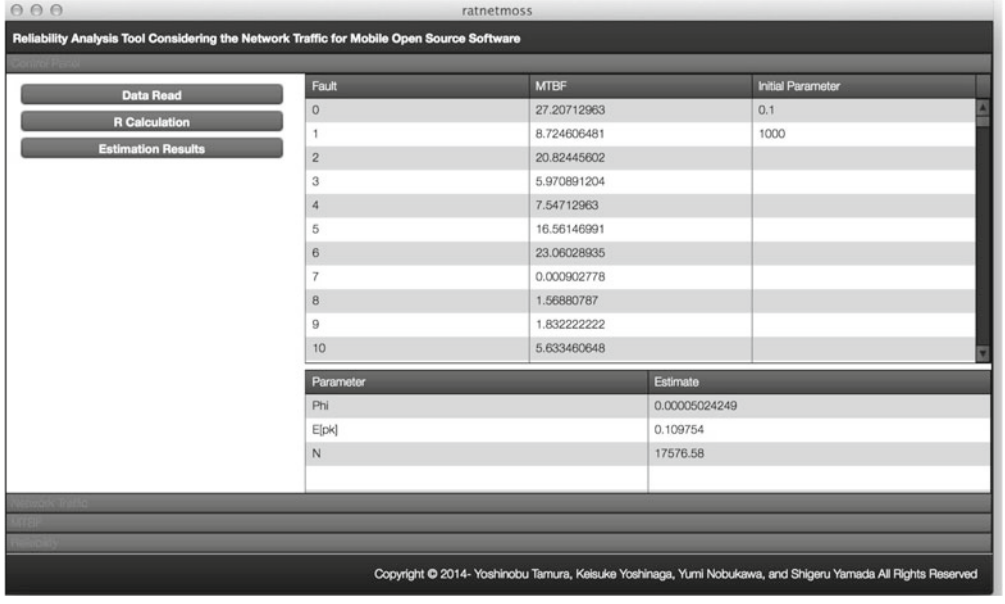

Fig. 12.19 The main screen of the developed AIR application in case of Firefox OS (version 1.2)

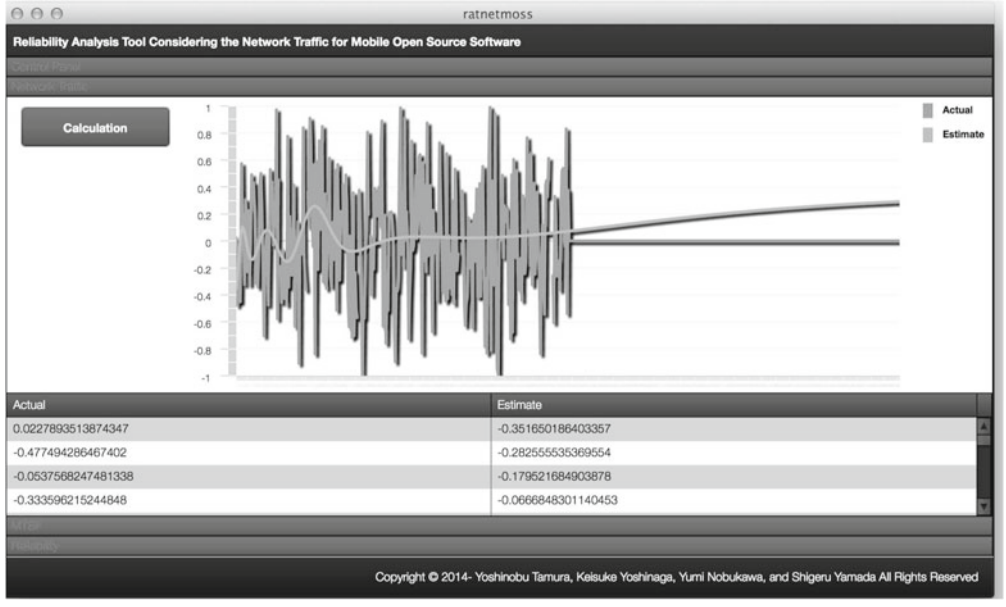

Fig. 12.20 The estimated network traffic density in case of Firefox OS (version 1.2)

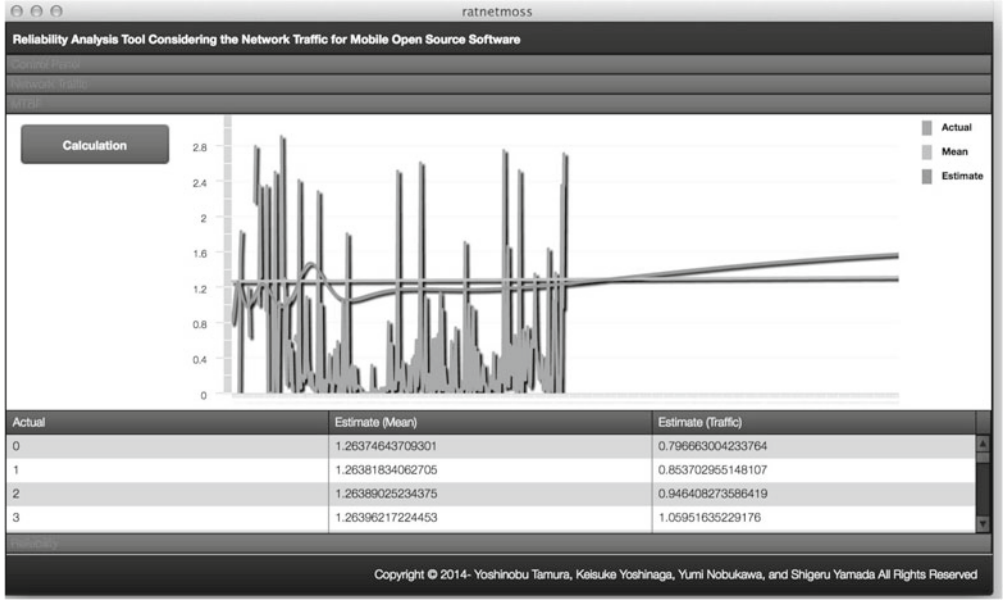

Fig. 12.21 The estimated MTBF in case of Firefox OS (version 1.2)

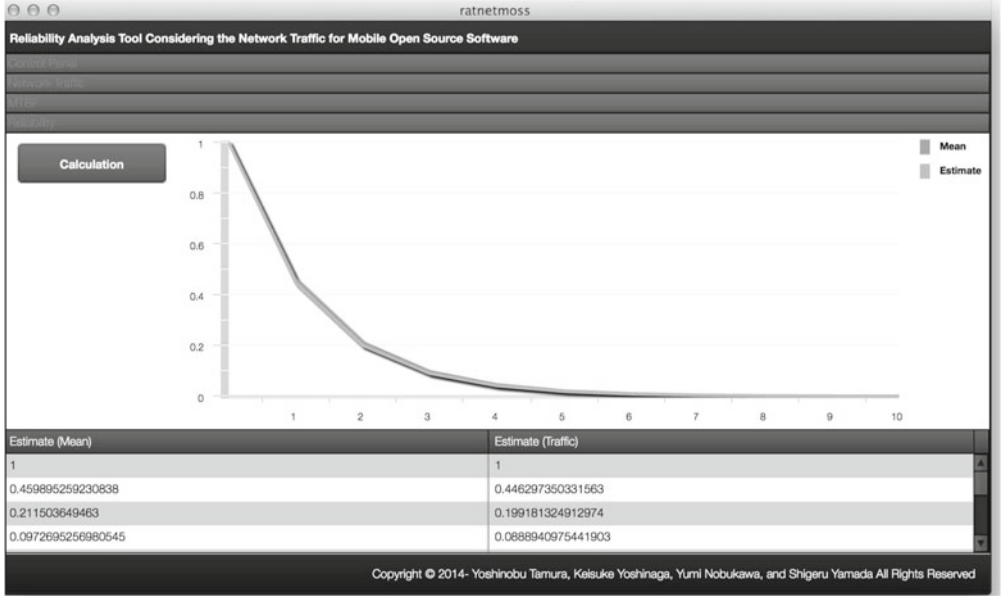

Fig. 12.22 The estimated software reliability in case of Firefox OS (version 1.2)

Finally, we have focused on the reliability for mobile clouds environment. The developed AIR application for reliability analysis in mobile clouds will be useful as a software tool of reliability analysis to solve the quality problems in mobile clouds. Moreover, it is useful for software managers to understand the debugging process and network traffic progress in operation phase of mobile clouds.

References

1. Y. Tamura, S. Yamada, Reliability and portability assessment tool based on hazard rates for an embedded open source software, J. Softw. 9(10) (Academy Publisher, 2014), pp. 2546–2556
2. Open Handset Alliance, Android. [Online]. http://www.android.com/
3. E. Andersen, BusyBox. [Online]. http://www.busybox.net/
4. Y. Tamura, S. Yamada, Reliability analysis tool based on stochastic differential equation models for an open source solution. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**(6), 78–85 (2013)
5. Flex.org–Adobe Flex Developer Resource, Adobe Systems Incorporated. http://flex.org/
6. The Apache HTTP Server Project, The Apache Software Foundation. http://httpd.apache.org/
7. Apache Tomcat, The Apache Software Foundation. [Online]. http://tomcat.apache.org/
8. MySQL, Oracle Corporation and/or its affiliates. [Online]. http://www.mysql.com/
9. Y. Tamura, S. Yamada, AIR application for reliability analysis considering debugging process and network traffic in mobile clouds, J. Simul. Model. Pract. Theory, (Elsevier B.V.). doi:10.1016/j.simpat.2014.03.010, Accessed 24 Apr 2014
10. Firefox OS, Marketplace, Android–Partners–mozilla.org, Mozilla Foundation. [Online]. http://www.mozilla.org/firefoxos/

Chapter 13
Exercises

This chapter shows several exercises for understanding the reliability assessment measures for OSS reliability measurement and assessment, e.g., MTBF, predicted relative error, and remaining faults. The following problems will be useful for software managers to evaluate OSS quality/reliability.

13.1 Exercise 1

Figure 13.1 shows the estimated MTBF in Eq. (10.4) based on the hazard rate model. Discuss the reliability trend of OSS in Fig. 13.1.

Brief Solution

It is important for software managers to understand the trends of reliability growth or regression. Figure 13.1 tends to show the trend of reliability growth because the estimated MTBF becomes large with the operating procedures go on.

13.2 Exercise 2

Figure 13.2 shows the estimated predicted relative error in Eq. (8.17). Discuss the goodness of fit of the estimated each model in Fig. 13.2.

Brief Solution

As shown in Fig. 12.13, the variation of all modeling estimations becomes stable when the debugging progress ratio exceeds 50 %.

13.3 Exercise 3

Figure 13.3 shows the estimated sample path of stochastic differential equation model in Eq. (9.19). Discuss the stability of the estimated sample paths in Fig. 13.3.

S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*,
Springer Series in Reliability Engineering, DOI 10.1007/978-3-319-31818-9_13

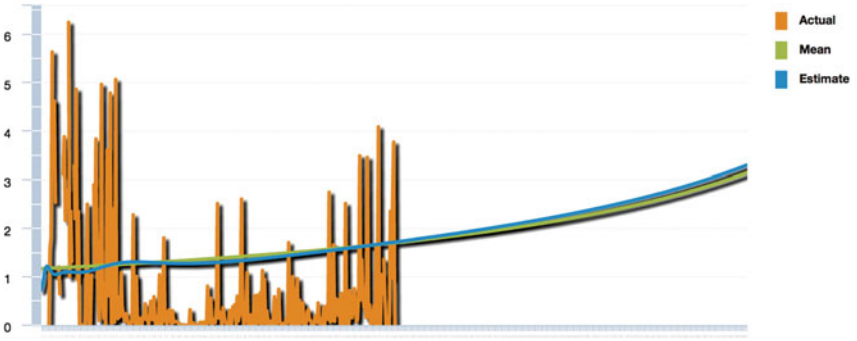

Fig. 13.1 The estimated MTBF based on the hazard rate model

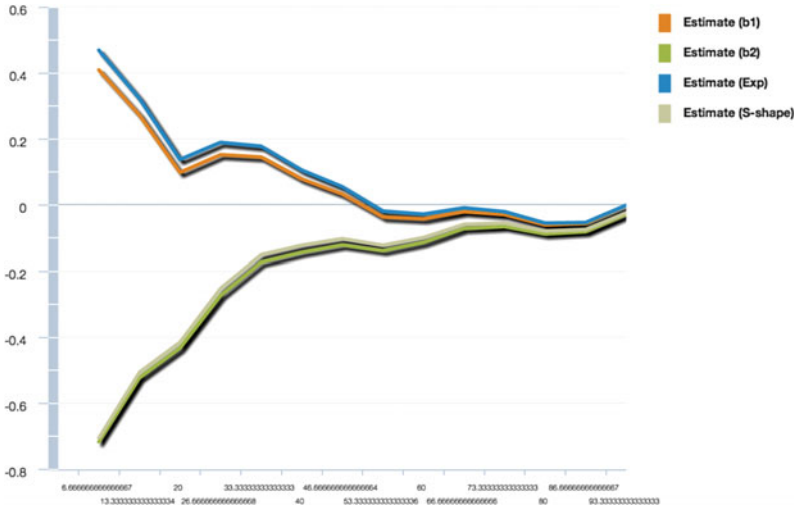

Fig. 13.2 The estimated predicted relative error

Brief Solution

Figure 12.11 means that the complexity of software component collision is large in the early debugging-process of open source solution.

13.4 Exercise 4

Figure 13.4 shows the estimated numbers of remaining faults in Eq. (9.22). Discuss the remaining faults on 15 days for each model in Fig. 13.4.

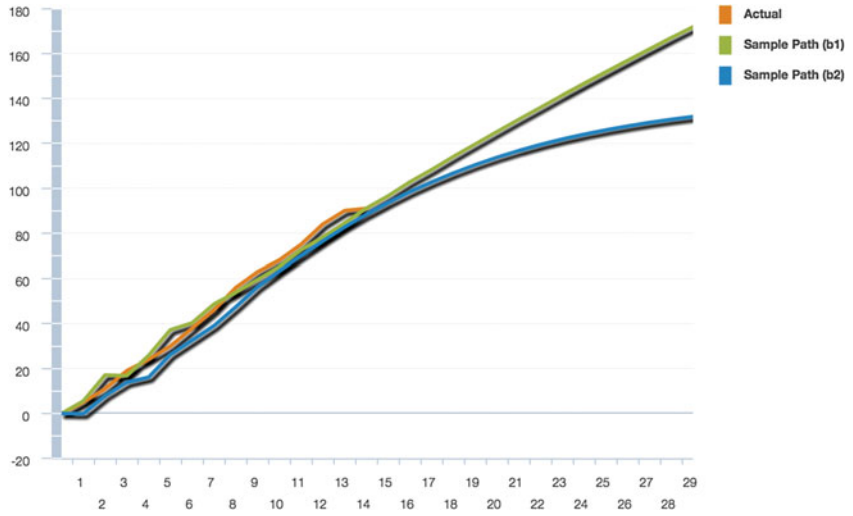

Fig. 13.3 The estimated sample path of stochastic differential equation model

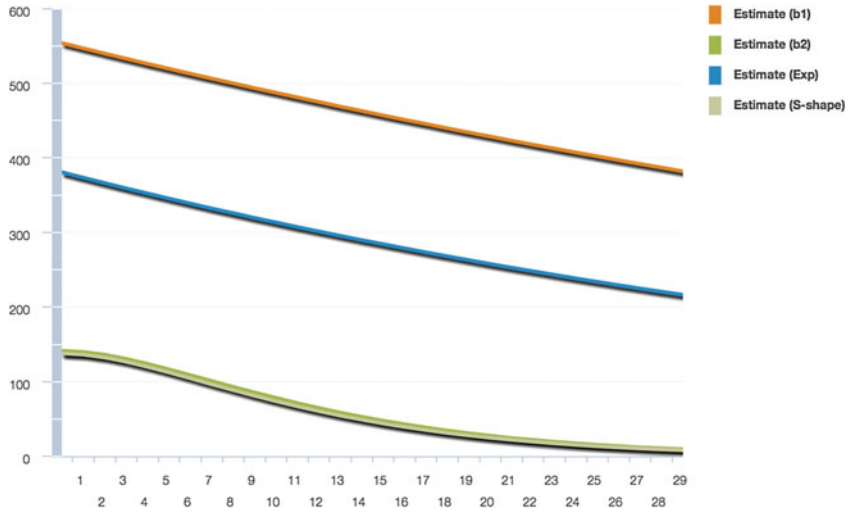

Fig. 13.4 The estimated remaining faults

Brief Solution

In Fig. 13.1, the estimated numbers of remaining faults is about 50 faults in case of "Estimate (S-shape)" on 15 days.