Component-oriented Reliability Analysis and Optimal Version-upgrade Problems for Open Source Software

Yoshinobu Tamura
Department of Computer Science, Faculty of Applied Information Science,
Hiroshima Institute of Technology, Japan
Email: tam@cc.it-hiroshima.ac.jp
Shigeru Yamada
Department of Management of Social Systems and Civil Engineering, Graduate School of Engineering,
Tottori Univerisity, Japan
Email: yamada@sse.tottori-u.ac.jp

***Abstract*— The current 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. Especially, OSS (Open Source Software) systems which serve as key components of critical infrastructures in our society are still ever-expanding now. However, poor handling of quality attainment and customer support prohibit the progress of OSS. We focus on the problems in low software quality that prohibit the progress of OSS.**

In case of considering the effect of the debugging process on an entire system in the development of a method of reliability assessment for open source project, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of each component, skill of fault reporter, and so on. In order to consider the effect of each software component on the reliability of an entire system under such OSS development, we propose a new approach to software reliability assessment by creating a fusion of neural networks and the software reliability growth models. Also, it has been necessary to manage the software development process in terms of reliability, effort, and version-upgrade time. In this paper, we find the optimal version-upgrade time based on the total expected software maintenance effort by using our software reliability growth models.

***Index Terms*— open source software, reliability, modeling, optimal version-upgrade time**

I. INTRODUCTION

The distributed development paradigm based on the object-oriented design and analysis will rapidly grow in the future, because the methodology is expected as a very effective approach to improve software quality and productivity. As a result of the technological progress, the social influence of software for business systems and the development effort of software systems are on the increase. Also, the 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 [1]. The current software development environment has changed into new 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 frequently applied as server use, instead of client use. Such OSS systems which serve as key components of critical infrastructures in the society are still ever-expanding now.

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 such open source projects includes GNU/Linux operating system[1], Apache Web server, and so on[2]. In this paper, we focus on OSS developed by using network computing technologies.

Software reliability growth models (SRGM's) [2] have been applied to assess the reliability for quality management and testing-progress control of software development. On the other hand, the effective testing management method for new distributed development paradigms as typified by the open source project has only a few presented [3]–[6]. In case of considering the effect of the debugging process on an entire system in the development of a method of reliability assessment for the OSS, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of each component, skill of fault reporter, and so on.

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

In order to consider the effect of each software component on the reliability of an entire system under such open source project, we apply a neural network [7]. Moreover, we discuss SRGM's which can evaluate the quality of OSS quantitatively. Also, we analyze actual software fault-count data to show numerical illustrations

[1]Linux is a Registered Trademark of Linus Torvalds.
[2]Other company, product, or service names may be trademarks or service marks of others.

of software reliability assessment for the open source project. Furthermore, we find the optimal version-upgrade time based on the total expected software maintenance effort.

II. RELIABILITY ASSESSMENT FOR EACH COMPONENT

A. Interaction among Software Components

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

We have proposed several software reliability assessment methods [8]–[10] based on AHP (Analytic Hierarchy Process) [11] for the OSS. However, it is difficult for the conventional software reliability assessment method based on AHP to estimate the weight parameter (i.e., importance factor) for each component. Because the AHP method requires the software developer's intention of decision-making in order to decide the evaluation criteria of the AHP, i.e., the software reliability assessment method based on the AHP is OSS developers-oriented.

In this paper, we propose a reliability assessment method based on the neural network in terms of estimating the effect of each component on the entire system in a complex situation. Especially, we consider that our method based on neural network is useful for OSS users to assess the software reliability by using the only data sets in bug tracking system on the website. Also, we can apply the importance level of faults detected during testing of each component, the size of component, the skill of fault reporter and so on, to the input data of neural network.

B. Weight Parameter for Each Component Based on Neural Network

In case of considering the effect of each component on the reliability entire system, it is necessary to grasp the size of each component, the skill of fault reporter, the state of error correction, the development time of component, the number of paths between components, and so on. In this paper, we apply the neural network to the estimation of interaction among software components in order to comprehend the internal state of OSS. We estimate the weight parameter for each component from input-output rules of neural network.

In this paper, we apply the structure of 3-layered neural networks. We assume that $w_{ij}^1(i = 1, 2, \cdots, I; j = 1, 2, \cdots, J)$ are the connection weights from i-th unit on the sensory layer to j-th unit on the association layer, and $w_{jk}^2(j = 1, 2, \cdots, J; k = 1, 2, \cdots, K)$ represent the connection weights from j-th unit on the association layer to k-th unit on the response layer. Moreover, $x_i(i = 1, 2, \cdots, I)$ are the normalized input values of i-th unit on the sensory layer, and $y_k(k = 1, 2, \cdots, K)$ represent the output values. We apply the number of critical(fatal) faults, the number of faults detected in specific operating system, the number of fault repairers, and the number of fault reporters, to the input values $x_i(i = 1, 2, \cdots, I)$.

The input-output rules of each unit on each layer are given by

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

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

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

$$f(x) = \frac{1}{1 + e^{-\theta x}}, \tag{3}$$

where θ is the gain of sigmoid function. We apply the multi-layered neural networks by back-propagation in order to learn the interaction among software components. We define the error function by the following equations:

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2, \tag{4}$$

where $d_k(k = 1, 2, \cdots, K)$ are the target input values for the output values. We apply the normalized values of the total number of faults for each component to the target input values $d_k(k = 1, 2, \cdots, K)$ for the output values, i.e., we consider the estimation and prediction model in which the property of the interaction among software components accumulates on the connection weights of neural networks. The connection weights w_{ij}^1 and w_{jk}^2 are estimated by using the following equations:

$$w_{jk}^2(\sigma + 1) = w_{jk}^2(\sigma) + \epsilon(y_k - d_k) \cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma) h_j\right) h_j, \tag{5}$$

$$w_{ij}^1(\sigma + 1) = w_{ij}^1(\sigma) + \epsilon \sum_{k=1}^{K}(y_k - d_k) \cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma) h_j\right) \cdot w_{ij}^1(\sigma) \cdot f'\left(\sum_{i=1}^{I} w_{ij}^1(\sigma) x_i\right) x_i. \tag{6}$$

In Eqs. (5) and (6), σ is the update cycle, and ϵ the learning parameter. By using the connection weights estimated from Eqs. (5) and (6), we can obtain the total weight parameter p_i $(i = 1, 2, \cdots, n)$ which represents the level of importance for each component as follows:

$$p_i = \frac{y_i}{\sum_{i=1}^{K} y_i}. \tag{7}$$

III. RELIABILITY ASSESSMENT FOR ENTIRE SYSTEM

A. Extended Logarithmic Poisson Execution Time Model

1) Model description: Many SRGM's have been used as the conventional methods to assess software reliability for quality management and testing-process control of software development. Among others, nonhomogeneous Poisson process (NHPP) models have been discussed in many literatures since the NHPP models can be easily applied in the software development. In this section, we discuss NHPP models for analyzing software fault-detection count data. Considering stochastic characteristics associated with fault-detection procedures in the testing-phase, we treat $\{N(t), t \geq 0\}$ as a nonnegative counting process where random variable $N(t)$ means the cumulative number of faults detected up to testing-time t. The fault-detection process $\{N(t), t \geq 0\}$ is described as follows [2]:

$$\Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!}\exp[-H(t)] \quad (n = 0, 1, 2, \cdots). \tag{8}$$

In Eq. (8), $\Pr\{\mathrm{A}\}$ means the probability of event A, and $H(t)$ is called a mean value function which represents the expected cumulative number of faults detected in the time interval $(0, t]$.

The OSS has the characteristics of the susceptible to various operational environments. Therefore, it is different from the conventional software system 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.

As mentioned above, we apply the logarithmic Poisson execution time model based on the assumption that the number of detected faults tends to infinity as testing-time $t \to \infty$. Thus, we consider the following structure of the mean value function represented by $\mu(t)$ because an NHPP model is characterized by its mean value function [8]–[10]:

$$\mu(t) = \frac{1}{\theta - P}\ln[\lambda_0(\theta - P)t + 1] \quad (0 < \theta,\ 0 < \lambda_0,\ 0 < P < 1), \tag{9}$$

where λ_0 is the intensity of initial inherent failure, and θ the reduction rate of the failure intensity rate per inherent fault. Moreover, we assume that the parameter P in Eq. (9) represents the following average in terms of the parameter y_i estimated by the neural network: $P = \sum_{i=1}^{n} y_i/n$, where n represents the number of software components [8]–[10].

2) Parameter estimation: In this section, the method of parameter estimation for the logarithmic Poisson execution time model based on an NHPP is presented. Suppose that K data pairs $(t_k, y_k)(k = 1, 2, \cdots, K)$ are observed during the operational phase, where the total number of software failures observed in the time-interval $(0, t_k]$ is $y_k(k = 1, 2, \cdots, K)$. Then, the logarithmic likelihood function of the NHPP model with mean value function $\mu(t)$ in Eq. (9) is given by

$$\ln L = \sum_{k=1}^{K}(y_k - y_{k-1}) \cdot \ln[\mu(t_k) - \mu(t_{k-1})] - \mu(t_K) - \sum_{k=1}^{K}\ln[(y_k - y_{k-1})!]. \tag{10}$$

The maximum-likelihood estimates $\hat{\theta}$ and $\hat{\lambda}$ for the unknown parameters θ and λ can be obtained by solving the following simultaneous likelihood equations numerically:

$$\frac{\partial \ln L}{\partial \theta} = \frac{\partial \ln L}{\partial \lambda} = 0. \tag{11}$$

3) Software reliability assessment measures: We can give the following expressions as software reliability assessment measures derived from the NHPP model given by Eq.(9):

- *Instantaneous fault-detection rate*
  The instantaneous fault-detection rate can be defined as the intensity function which represents the number of faults detected per unit time. From Eq. (9), the instantaneous fault-detection rate is defined as follows:
  $$IR(t) = \frac{d\mu(t)}{dt}. \tag{12}$$
- *Instantaneous mean time between software failures*
  The instantanesous mean time between software failures ($\mathrm{MTBF_I}$) is given as follows:
  $$MTBF_I(t) = \frac{1}{d\mu(t)/dt}. \tag{13}$$
- *Cumulative mean time between software failures*
  The cumulative mean time between software failures ($\mathrm{MTBF_C}$) is given as follows:
  $$MTBF_C(t) = \frac{t}{\mu(t)}. \tag{14}$$

B. Stochastic Differential Equation Model

1) Model description: Let $S(t)$ be the cumulative number of faults detected in the OSS system at operational 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. Considering the characteristics of open source software development, the OSS developers report several related-faults when the OSS developers confirm the specific faults of bug tracking system, i.e., OSS developers can be OSS users. Therefore, we assume that the increase rate of $S(t)$ is proportional to the value $S(t)$ itself. Thus, under common assumptions for software reliability growth modeling, we consider the following linear differential equation:

$$\frac{dS(t)}{dt} = \lambda(t)S(t), \tag{15}$$

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

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

Therefore, we suppose that $\lambda(t)$ in Eq.(15) has the irregular fluctuation. That is, we extend Eq.(15) to the following stochastic differential equation [12], [13]:

$$\frac{dS(t)}{dt} = \{\lambda(t) + \sigma\gamma(t)\}\, S(t), \tag{16}$$

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

We extend Eq.(16) to the following stochastic differential equation of an Itô type:

$$dS(t) = \{\lambda(t) + \frac{1}{2}\sigma^2\}S(t)dt + \sigma S(t)dW(t), \tag{17}$$

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 has the following properties:

$$\begin{aligned} \Pr[W(0) = 0] &= 1, & (18)\\ \mathrm{E}[W(t)] &= 0, & (19)\\ \mathrm{E}[W(t)W(t')] &= \mathrm{Min}[t, t']. & (20) \end{aligned}$$

By using Itô's formula [12], [13], we can obtain the solution of Eq.(16) under the initial condition $S(0) = v$ as follows [14] :

$$S(t) = v \cdot \exp\left(\int_0^t \lambda(s)ds + \sigma W(t) \right), \tag{21}$$

where v is the number of detected faults for the previous software version. Using solution process $S(t)$ in Eq.(21), we can derive several software reliability measures.

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

$$\int_0^t \lambda(s)ds = \sum_{i=1}^{n} p_i(1 - \exp[-\alpha_i t]), \tag{22}$$

where α_i $(i = 1, 2, \cdots, n)$ is an acceleration parameter of the intensity of inherent software failures for the i-th fault importance level, p_i $(\sum_{i=1}^{n} p_i = 1)$ the weight parameter for the i-th fault importance level, and n the number of the applied fault importance levels. We can apply the S-shaped growth curve to Eq. (22) depending on the trends of fault importance level.

2) Method of maximum-likelihood: In this section, the estimation method of unknown parameters α and σ in Eq. (21) is presented. Let us denote the joint probability distribution function of the process $S(t)$ as

$$\begin{aligned} &P(t_1,\ y_1;\ t_2,\ y_2;\ \cdots;\ t_K,\ y_K)\\ &\equiv \Pr[S(t_1) \le y_1,\ \cdots,\ S(t_K) \le y_K | S(t_0) = v], \end{aligned} \tag{23}$$

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

$$\begin{aligned} &p(t_1,\ y_1;\ t_2,\ y_2;\ \cdots;\ t_K,\ y_K)\\ &\quad\equiv \frac{\partial^K P(t_1,\ y_1;\ t_2,\ y_2;\ \cdots;\ t_K,\ y_K)}{\partial y_1 \partial y_2 \cdots \partial y_K}. \end{aligned} \tag{24}$$

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

$$l = p(t_1,\ y_1;\ t_2,\ y_2;\ \cdots;\ t_K,\ y_K). \tag{25}$$

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

$$L = \log l. \tag{26}$$

The maximum-likelihood estimates α_i^* and σ^* are the values making L in Eq. (26) maximize. These can be obtained as the solutions of the following simultaneous likelihood equations [14] :

$$\frac{\partial L}{\partial \alpha_i} = \frac{\partial L}{\partial \sigma} = 0. \tag{27}$$

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

$$f(W(t)) = \frac{1}{\sqrt{2\pi t}} \exp\left\{ -\frac{W(t)^2}{2t} \right\}. \tag{28}$$

Information on the current number of detected faults in the system is important to estimate the situation of the progress on the 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.(21) as follows [14] :

$$\mathrm{E}[S(t)] = v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t \right), \tag{29}$$

$$\begin{aligned} \mathrm{Var}[S(t)] &= \mathrm{E}[\{S(t) - \mathrm{E}[S(t)]\}^2]\\ &= v^2 \cdot \exp\left(2\int_0^t \lambda(s)ds + \sigma^2 t \right)\\ &\cdot \left\{ \exp(\sigma^2 t) - 1 \right\}, \end{aligned} \tag{30}$$

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

From Eq. (29), we can confirm that the number of detected faults can not converge to a finite value as the following equation:

$$\lim_{t \to \infty} \mathrm{E}[S(t)] = \infty. \tag{31}$$

The 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 expected number of detected faults can not converge to a finite value [8]–[10].

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

$$MTBF_I(t) = \frac{1}{\mathrm{E}[\frac{dS(t)}{dt}]}. \tag{32}$$

Therefore, we have the following instantaneous MTBF:

$$\begin{aligned} MTBF_I(t) &= 1 \Big/ v\left(\lambda(t) + \frac{1}{2}\sigma^2\right) \\ &\cdot \quad \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right). \end{aligned} \tag{33}$$

Also, cumulative mean time between software failures (denoted by $\mathrm{MTBF_C}$) is approximately given by

$$MTBF_C(t) = \frac{t}{\mathrm{E}[S(t)]}. \tag{34}$$

Therefore, we have the following cumulative MTBF:

$$MTBF_C(t) = \frac{t}{v \cdot \exp\left(\int_0^t \lambda(s)ds + \frac{\sigma^2}{2}t\right)}. \tag{35}$$

Moreover, we can derive the following coefficient of variation from Eq. (21):

$$CV(t) \equiv \frac{\sqrt{\mathrm{Var}[S(t)]}}{\mathrm{E}[S(t)]}. \tag{36}$$

IV. OPTIMAL VERSION-UPGRADE PROBLEM

Recently, it becomes more difficult for software developers to produce highly-reliable software systems efficiently, because of the diversified and complicated software requirements. Thus, it has been necessary to control the software development process in terms of reliability, cost, and delivery time. On the other hand, the effective optimal software version-upgrade problem for OSS has only a few presented. It is very important in terms of software management that we decide for the optimal length of testing versions for OSS. We find the optimal testing-time based on the total expected software maintenance effort in this section. Several optimal software release problems considering host-concentrated software development process have been proposed by many researchers [15], [16]. However, optimal software release problems for OSS have not been proposed. Therefore, we formulate a maintenance effort model based on our two SRGM's proposed in Section III, and analyze the optimal release problem minimizing the total expected maintenance effort.

It is interesting for the software developers to predict and estimate the time when we should stop testing in order to develop a highly reliable software system efficiently. Hence, we discuss about the determination of software version-upgrade times minimizing the total expected software effort.

We define the following:

m_0: the fixing effort per fault during the test-version,
m_1: the maintenance effort per fault during the test-version,
m_2: the effort per time for fixing faults during the test-version.

Then, the expected software effort in the test-version of OSS for our two SRGM's can be formulated as:

$$E_1(t) = m_0 \cdot \mu(t), \tag{37}$$
$$E_1(t) = m_0 \cdot \mathrm{E}[S(t)]. \tag{38}$$

Also, the expected software maintenance effort after the release of general availability for our two SRGM's is represented as follows:

$$E_2(t) = m_1\{\mu(t_0) - \mu(t)\} + m_2 t, \tag{39}$$
$$E_2(t) = m_1\{\mathrm{E}[S(t_0)] - \mathrm{E}[S(t)]\} + m_2 t. \tag{40}$$

where, t_0 is the previous version-upgrade period.

Moreover, if the software components are added to entire system after the version-upgrade, the penalty effort is imposed. We define the penalty effort function as follows:

$$G(t) = (1-c)\exp\left[\frac{t-t_0}{v}\right], \tag{41}$$

where c is the ratio of the number of new components to entire system after version-upgrade, and v the number of prior version-upgrade.

Consequently, from Eqs.(37), (39), and (41), the total expected software effort is given by

$$E(t) = E_1(t) + E_2(t) + G(t). \tag{42}$$

The optimum version-upgrade time t^* is obtained by minimizing $E(t)$.

V. NUMERICAL ILLUSTRATIONS

We focus on the Fedora Core Linux [17] 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[3]. 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 paper are collected in the bug tracking system on the website of Fedora project in May 2007. Especially, we focus on the Kernel component of the Fedora Core Linux (FC7).

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

TABLE I.
THE ESTIMATED WEIGHT PARAMETERS IN FC7

Component Name	Weight parameters p_i
Kernel	p_1=0.882
Kernel-xen	p_2=0.069
Kernel-module-thinkpad	p_3=0.012
Kernel-pcmcia-cs	p_4=0.012
Kernel-utils	p_5=0.012
Kenrel-xen-2.6	p_6=0.012

A. Software Reliability Assessment Procedures

The procedures of reliability assessment in the proposed our method for OSS are shown as follows:

1. We processes the data file in terms of the data in bug-tracking system of the specified OSS for reliability assessment.
2. Using the data obtained from bug-tracking system, we process the data for input data.
3. We estimate the weight parameter p_i $(i = 1, 2, \cdots, n)$ for each component by using the neural network. Then, the parameter P in our model is obtained from Eq. (7).
3. Also, the unknown parameters included in our models are estimated by using the maximum-likelihood method.
4. Moreover, we show the number of detected faults, the instantaneous fault-detection rate, and the cumulative MTBF as software reliability assessment measures, and the predicted relative error.
5. Finally, we estimate the total expected software effort. Then, we can confirm the optimal version-upgrade time [18].

B. Level of Importance for Each Component

Estimating the weight parameter in terms of the reliability by using the neural network, the input data sets are the importance level of faults detected for each component (Critical), the platform (All), the fault repairer (Assigned to), and the fault reporter (Reporter).

The estimated results of weight parameter $p_i (i = 1, 2, \cdots, 6)$ for the Kernel of FC7 based on the neural network in Section II-B are shown in Tables I. From Table I, we can grasp the level of importance in terms of reliability for each component.

C. Reliability Assessment for Entire System

On the presupposition that the weight parameters for each component are estimated by using the neural network, we show numerical examples for reliability assessment of FC7. The estimated numbers of detected faults of FC7 in Eq. (9), $\widehat{\mu}(t)$ by using our models are shown in Figure 1.

D. Goodness-of-fit Comparison

We compare the goodness-of-fit of the proposed SRGM's for OSS. We adopt the value of the Mean Square Error (MSE) as comparison criteria of goodness-of-fit.

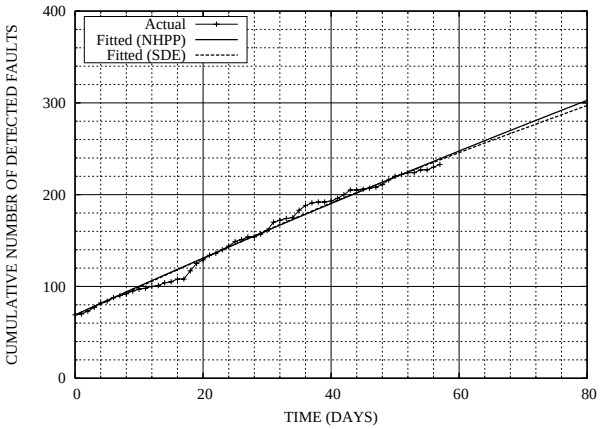

Figure 1. The estimated cumulative number of detected faults.

TABLE II.
COMPARISON OF THE MSE FOR THE CUMULATIVE NUMBER OF DETECTED FAULTS OF FIREFOX WEB BROWSER.

Compared methods	MSE
NHPP model	565.52
SDE model	457.48

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,

$$\text{MSE} = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2, \tag{43}$$

where $\hat{y}_k$ in Eq. (43) is obtained from estimated $\widehat{\mu}(t_k)$, and $\widehat{\mathrm{E}}[S(t_k)]$, $(k = 1, 2, \cdots, K)$. The MSE indicates that the selected method and model fit better to the observed data as the MSE becomes small.

Table II shows the result of goodness-of-fit comparison in terms of the MSE for the observed data of Firefox Web browser [19]. We found that the proposed method based on chaos theory fits better than the other SRGM's with respect to MSE.

Moreover, we show the estimated cumulative number of detected faults by using proposed SRGM's for OSS in Figure 2.

Furthermore, Figure 3 shows the behaviors of the predicted relative error for NHPP model and SDE model, respectively. As shown in Figure 3, the estimate by NHPP model becomes stable when the progress ratio (t_e/t_q) exceeds 60%.

E. Optimal Version-upgrade Time

In this section, we show several numerical examples of FC7 based on the optimal version-upgrade problems which are discussed in the section IV. Figure 4 is shown the estimated total expected software effort by using our models. From Figure 4 in case of the extended logarithmic Poisson execution time model, we find that the optimum version-upgrade time is derived as $t^* = 133$ days from

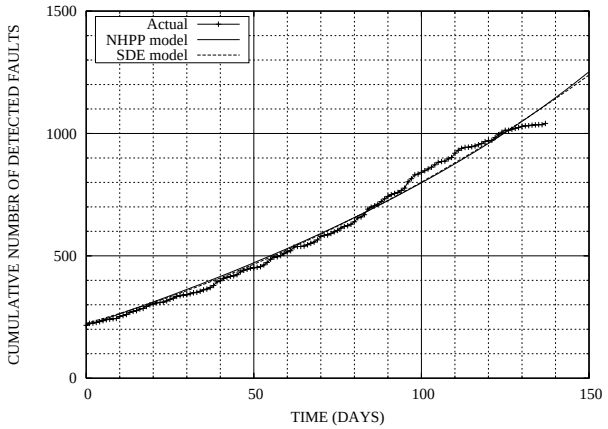

Figure 2. The estimated cumulative number of detected faults for each SRGM.

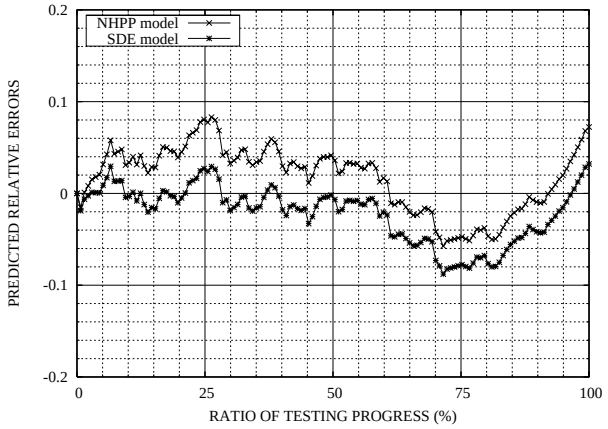

Figure 3. Estimated predicted relative errors for each SRGM.

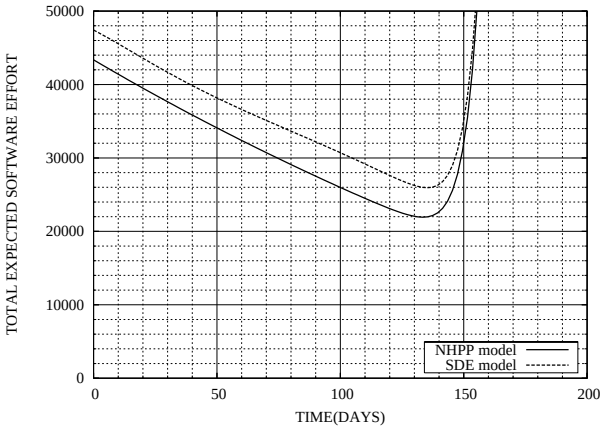

Figure 4. The estimated total expected software effort for our models

Figure 4. Then, the total expected software effort is 21498 man-days effort. On the other hand, from Figure 4 in case of the stochastic differential equation model, we find that the optimum version-upgrade time is derived as $t^* = 139$ days from Figure 4. Then, the total expected software effort is 25019 man-days effort.

VI. CONCLUDING REMARKS

In this paper, we have focused on the Fedora Core Linux operating system which is known as the OSS, and discussed the method of reliability assessment for the OSS developed under on an open source project. Especially, we have applied on the neural network in order to consider the effect of each software component on the reliability of an entire system under such open source development paradigm. By using the neural network, we have proposed the method of reliability assessment incorporating the interaction among software components. The neural network and NHPP model applied in this paper have simple structures. Therefore, we can easily apply our method to actual open source software by rote.

In case of considering the effect of debugging process on an entire system in the development of software reliability assessment methods for open source projects, it is necessary to grasp the deeply-intertwined factors. In this paper, we have shown that our method can grasp such deeply-intertwined factors by using the neural network. Especially, we consider that our method based on neural network is useful for OSS users to assess the software reliability by using the data sets in bug tracking system on the website [20].

Moreover, it has been necessary to control the software development process in terms of reliability, effort, and version-upgrade time for OSS. We have formulated the maintenance effort model based on our SRGM's and analyzed the optimal release problem minimizing the total expected maintenance effort. Also, we have estimated the optimum version-upgrade time.

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 quantitative reliability assessment incorporating the importance of each component for an entire system.

ACKNOWLEDGMENTS

This work was supported in part by the Grant-in-Aid for Scientific Research (C), Grant No. 18510124 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

REFERENCES

[1] A. Umar, *Distributed Computing and Client–Server Systems*, Prentice Hall, New Jersey, 1993.
[2] S. Yamada, *Software Reliability Models: Fundamentals and Applications* (in Japanese), JUSE Press, Tokyo, 1994.
[3] A. MacCormack, J. Rusnak, and C.Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Science*, vol. 52, no. 7, 2006, pp. 1015–1030.
[4] G. Kuk, "Strategic interaction and knowledge sharing in the KDE developer mailing list," *Management Science*, vol. 52, no. 7, 2006, pp. 1031–1042.

[5] Y. Zhoum, J. Davis, "Open source software reliability model: an empirical approach," *Proceedings of the Workshop on Open Source Software Engineering (WOSSE)*, vol. 30, no. 4, 2005, pp. 67–72.

[6] P. Li, M. Shaw, J. Herbsleb, B. Ray and P. Santhanam, "Empirical evaluation of defect projection models for widely-deployed production software systems," *Proceedings of the 12th International Symposium on the Foundations of Software Engineering (FSE-12)*, 2004, pp. 263–272.

[7] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Trans. Neural Networks.*, vol. 1, June 1990, pp. 239-242.

[8] Y. Tamura, S. Yamada and M. Kimura, "Reliability assessment method based on logarithmic Poisson execution time model for open source project," *Proceedings of the Second IASTED International Multi-Conference on Automation, Control, and Information Technology*, Novosibirsk, Russia, June 20–24, 2005, pp. 54–59.

[9] Y. Tamura and S. Yamada, "Comparison of software reliability assessment methods for open source software," *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005)–Volume II*, Fukuoka, Japan, July 20–22, 2005, pp. 488–492.

[10] Y. Tamura and S. Yamada, "Validation of an OSS reliability assessment method based on ANP and SRGM's," *Proceedings of the International Workshop on Recent Advances in Stochastic Operations Research*, Canmore, Canada, August 25–26, 2005, pp. 273–280.

[11] T. Satty, *The Analytic Hierarchy Process*, McGraw–Hill, New York, 1980.

[12] L. Arnold, *Stochastic Differential Equations–Theory and Applications*, John Wiley & Sons, New York, 1974.

[13] E. Wong, *Stochastic Processes in Information and Systems*, McGraw–Hill, New York, 1971.

[14] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, "Software reliability measurement and assessment with stochastic differential equations," *IEICE Trans. Fundamentals*, vol. E77-A, no. 1, pp. 109-116, Jan. 1994.

[15] S. Yamada, H. Narihisa, and S. Osaki, "Optimal release policies for a software system with a scheduled software delivery time," *Intern. J, Systems Science*, vol. 15, no. 8, Aug. 1984, pp. 905–914.

[16] S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements," *European J. Operational Research*, vol. 31, no. 1, July 1987, pp. 46–51.

[17] Fedora Project, sponsored by Red Hat. [Online]. Available: http://fedora.redhat.com/

[18] Y. Tamura and S. Yamada, "Software reliability assessment and optimal version-upgrade problem for open source software," *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics*, Montreal, Canada, Oct. 7–10, 2007, pp. 1333–1338.

[19] Mozilla.org, Mozilla Foundation. [Online]. Available: http://www.mozilla.org/

[20] Y. Tamura, K. Hadatsuki, S. Yamada, and M. Kimura, "Development of a user-oriented reliability assessment tool for open source software (in Japanese)," *Proceedings of the Linux Conference 2006*, vol. 4. [Online]. Available: http://lc.linux.or.jp/lc2006/

Yoshinobu Tamura received the B.S.E., M.S., and Ph.D. degrees from Tottori University in 1998, 2000, and 2003, respectively. From 2003 to 2006, he was a Research Assistant at Tottori University of Environmental Studies. Since 2006, he has been working as a Lecturer at the Faculty of Applied Information Science of Hiroshima Institute of Technology, Hiroshima, Japan. His research interests include reliability assessment for open source software. He is a regular member of the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, the Operations Research Society of Japan, and the Society of Project Management of Japan, and the IEEE.

Shigeru Yamada received the B.S.E., M.S., and Ph.D. degrees from Hiroshima University in 1975, 1977, and 1985, respectively. From 1977 to 1980, he worked at the Quality Assurance Department of Nippondenso Co., Aichi Prefecture, Japan. From 1983 to 1988, he was an assistant professor of Okayama University of Science, Okayama, Japan. From 1988 to 1993, he was an associate professor at the Faculty of Engineering of Hiroshima University, Japan. Since 1993, he has been working as a professor at the Faculty of Engineering of Tottori University, Tottori-shi, Japan. He has published numerous technical papers in the area of software reliability models, project management, reliability engineering, and quality control. He has authored several books entitled *Software Reliability: Theory and Practical Application* (Soft Research Center, Tokyo, 1990), *Introduction to Software Management Model* (Kyoritsu Shuppan, 1993), *Software Reliability Models: Fundamentals and Applications* (JUSE, Tokyo, 1994), *Statistical Quality Control for TQM* (Corona Publishing, Tokyo, 1998), *Software Reliability: Model, Tools, Management* (The Society of Project Management, 2004), and *Quality-Oriented Software Management* (Morikita Shuppan, 2007). Dr. Yamada received the Best Author Award from the Information Processing Society of Japan in 1992, the TELECOM System Technology Award from the Telecommunications Advancement Foundation in 1993, the Paper Award from the Reliability Engineering Association of Japan in 1999, the International Leadership Award in Reliability Engg. Research from the ICQRIT/SREQOM in 2003, the Best Paper Award from the Society of Project Management in 2006, and the Leadership Award from the ISSAT in 2007. He is a regular member of the IEICE, the Information Processing Society of Japan, the Operations Research Society of Japan, the Japan SIAM, the Reliability Engineering Association of Japan, Japan Industrial Management Association, the Japanese Society for Quality Control, the Society of Project Management of Japan, and the IEEE.