A Novel Software Reliability Assessment Approach based on Neural Network in Network Environment

Changjie Ma, Guochang Gu, Jing Zhao
College of Computer Science and Technology, Harbin Engineering University, Harbin, 150001, China
changjiema234@163.com

Abstract
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 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.

Keywords: Software Reliability Assessment, Neural Network, Network Environment

1. Introduction
All over the world people can gain 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. Such 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 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, 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 system1, Apache Web server, and so on2. 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], [4]. 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 [5]. Moreover, we analyze actual software fault count data to show numerical illustrations of software reliability assessment for the open source project. Furthermore, we find the optimal total version-upgrade time based on the total expected software maintenance effort.
2. Reliability Assessment For Each Component

2.1 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 [6], [7], [8] based on AHP (Analytic Hierarchy Process) [9] for the OSS. However, it is difficult for the conventional software reliability assessment method based on AHP to estimate the weight parameter 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.

2.2 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 path 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 in Fig. 1, where 

\[ w_{ij} (i = 1, 2, \ldots, I, \ j = 1, 2, \ldots, J) \]

are the connection weights from i-th unit on the sensory layer to j-th unit on the association layer, and 

\[ w_{jk} (j = 1, 2, \ldots, J, \ k = 1, 2, \ldots, K) \]

represent the connection weights from j-th unit on the association layer to k-th unit on the response layer. Moreover, in Fig. 1, 

\[ x_i (i = 1, 2, \ldots, I) \]

are the normalized input values of i-th unit on the sensory layer, and 

\[ y_k (k = 1, 2, \ldots, K) \]

represent the output values. We apply the number of critical (fatal) faults, the number of fault detected in specific operating system, fault repairer, and fault reporter, to the input values \[ x_i (i = 1, 2, \ldots, I) \].
The input-output rules of each unit on each layer are given by

\[ h_i = f \left( \sum_{i=1}^{J} w^1_{ij} x_i \right) \]  
(1)

\[ y^1_k = f \left( \sum_{j=1}^{I} w^2_{kj} h_j \right) \]  
(2)

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

\[ f(x) = \frac{1}{1 + e^{-\theta x}} \]  
(3)

where \( \theta \) is the gain of sigmoid function. We apply the multilayered neural networks by backpropagation in order to learn the interaction among software components. We define the error function by the following equations:

\[ E = \frac{1}{2} \sum_{k=1}^{K} (y^1_k - d_k)^2 \]  
(4)

where \( d_k \) (\( k = 1, 2, \ldots, K \)) 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 \( d_k \) (\( k = 1, 2, \ldots, K \)) 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 \( w^1_{ij} \) and \( w^2_{ij} \) are estimated by using the following equation:

\[ w^2_{jk} (\sigma +1) = w^2_{jk} (\sigma) + \varepsilon (y^1_k - d_k) f' \left( \sum_{j=1}^{J} w^2_{jk} (\sigma) h_j \right) h_j \]  
(5)

\[ w^1_{ij} (\sigma +1) = w^1_{ij} (\sigma) + \varepsilon \sum_{k=1}^{K} (y^1_k - d_k) f' \left( \sum_{j=1}^{J} w^2_{jk} (\sigma) h_j \right) \left[ \sum_{i=1}^{I} w^1_{ij} (\sigma) x_i \right] x_i \]  
(6)
In Eqs. (5) and (6), $\sigma$ is the update cycle, and $\epsilon$ is the learning parameter. By using the connection weights estimated from Eqs. (5) and (6), we can obtain the total weight parameter $p_i$ ($i = 1, 2, \ldots, n$) which represents the level of importance for each component as follows:

$$p_i = \frac{y_i}{\sum_{i=1}^{n} y_i}$$

(7)

3. Reliability Assessment For Entire System

3.1 Reliability Assessment Based on Conventional SRGM’s

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]:

$$\Pr\{N(t) = n\} = \frac{(H(t))^n}{n!} \exp[-H(t)](n = 0, 1, 2, \ldots)$$

(8)

In Eq. (8), $\Pr\{A\}$ 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 $(0, t]$.

3.2 Extended Logarithmic Poisson Execution Time Model

The operating environment of 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 continue 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 $\mu(t)$ because an NHPP model is characterized by its mean value function[6], [7], [8]:

$$\mu(t) = \frac{1}{\theta - P} \ln \left[ \frac{\lambda_0}{\theta - P} (\theta - P)^{t+1} \right] (0 < \theta, 0 < \lambda_0, 0 < P < 1)$$

(9)

where $\lambda_0$ is the intensity of initial inherent failure, and $\theta$ 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[6], [7], [8].

3.3 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 $(t_k, y_k)(k = 1, 2, \ldots, K)$ are observed during the operational phase, where the total number of software failures observed in the time-interval $(0, t_k]$ is $y_k(k = 1, 2, \ldots, K)$. Then, the logarithmic likelihood function of the NHPP model with mean value function $\mu$ of Eq. (9) is given by
The maximum-likelihood estimates \( \hat{\theta} \) and \( \hat{\lambda} \) for the unknown parameters \( \theta \) and \( \lambda \) can be obtained by solving the following simultaneous likelihood equations numerically:

\[
\frac{\partial \ln L}{\partial \theta} = \frac{\partial \ln L}{\partial \lambda} = 0
\]

### 3.4 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:

  \[
  IR(t) = \frac{d\mu(t)}{dt}
  \]

- **Instantaneous mean time between software failures**
  
  The instantaneous mean time between software failures (MTBFI) is given as follows:

  \[
  MTBF_I(t) = \frac{1}{d\mu(t)/dt}
  \]

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

  \[
  MTBF_C(t) = \frac{t}{\mu(t)}
  \]

### 4. Optimal Version-Upgrade Problem

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. 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[10], [11]. However, optimal software release problems for OSS have not been proposed. Therefore, we formulate a maintenance effort model based on our SRGM 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 can be formulated as:

\[
E(t) = m_0 \mu(t)
\]

Also, the expected software maintenance effort after the release of general availability is represented as follows:
where, \( t_0 \) is the previous version-upgrade period.

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

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

(17)

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

Consequently, from Eqs. (15), (16), and (17), the total expected software effort is given by

\[
E(t) = E_1(t) + E_2(t) + G(t)
\]  

(18)

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

5. Numerical Illustrations

We focus on the Fedora Core Linux[12] which is one of the operating systems 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. 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 the Fedora project in October 2006. Especially, we focus on the Kernel component of the Fedora Core Linux (FC6).

5.1 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 parameters \( p_i \) (\( i = 1, 2, \ldots, n \)) for each component by using the neural network. Then, the parameter \( P \) in our model is obtained from Eq. (7).
4. Also, the unknown parameters \( \theta \) and \( \lambda_0 \) included in our model are estimated by using the maximum-likelihood method.
5. 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.
6. Finally, we estimate the total expected software effort. Then, we can confirm the optimal version-upgrade time.

5.2 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).

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Weight parameters ( p_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>( p_1 = 0.886 )</td>
</tr>
<tr>
<td>Kernel-xen</td>
<td>( p_2 = 0.074 )</td>
</tr>
<tr>
<td>Kernel-module-thinkpad</td>
<td>( p_3 = 0.013 )</td>
</tr>
<tr>
<td>Kernel-pcmcia-cs</td>
<td>( p_4 = 0.013 )</td>
</tr>
<tr>
<td>Kernel-utils</td>
<td>( p_5 = 0.013 )</td>
</tr>
</tbody>
</table>
The estimated results of weight parameter $p(i = 1, 2, \ldots, 5)$ for the Kernel of FC6 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.

### 5.3 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 FC6. The estimated numbers of detected faults of FC6 in Eq. (9), $\hat{\mu}(t)$ are shown in Fig. 2. From Fig. 2, we can find that the number of detected faults at the 130 days is about 100 faults. Moreover, the estimated instantaneous fault detection rates for the FC6 is shown in Fig. 3. Furthermore, the estimated instantaneous MTBF and cumulative MTBF are also plotted in Figs. 4 and 5, respectively. From Figs. 4 and 5, the estimated instantaneous MTBF and cumulative MTBF slowly grow as the time elapses after the evaluated version of FC6 has been released.

**Fig. 2.** The estimated number of detected faults

**Fig. 3.** The estimated instantaneous fault-detection rate

**Fig. 4.** The estimated instantaneous MTBF

**Fig. 5.** The estimated cumulative MTBF

### 5.4 Optimal Version-upgrade Time

In this section, we show several numerical examples based on the optimal version-upgrade problems which are discussed in the section IV. Fig. 6 is shown the estimated total expected software effort. From Fig. 6, we find that the optimum version-upgrade time is derived as $t^* = 149$ days from Fig. 6. Then, the total expected software effort is 2205.07.

Moreover, we have verified that our effort model can be applied to estimate version-upgrade time of the OSS. We show some behaviors of our effort model if we change the parameters $v$ and $c$ which are the number of prior version-upgrade and the ratio of the number of new components to entire system after version-upgrade. In addition to the case of $v = 5$ and $c = 0$, we represent the estimated total expected software effort with changing the value of the parameter $v$ and $c$ at regular intervals are illustrated in Figs. 7 and 8, respectively. From Figs. 7 and 8, we have found that the total expected software effort decreases with the parameter $v$ and $c$ growth. On the other hand, the optimal version-upgrade time increases with the parameter $v$ and $c$ growth.
6. Concluding Remarks

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 user to assess the software reliability by
using the data sets in bug tracking system on the website[13].

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

7. References

designs: An empirical study of open source and proprietary code,” Management Science, vol. 52,
no. 7, pp. 1015–1030, 2006
Trans. Neural Networks., vol. 1, pp. 239-242, June 1990


