# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities




WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

## Interested in publishing with us?
## Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



13

# Reliability Analysis Methods for an Embedded Open Source Software

Yoshinobu Tamura † and Shigeru Yamada ‡
*Graduate School of Science and Engineering, Yamaguchi University†*
*Graduate School of Engineering, Tottori University‡*

## 1. Introduction

Many software systems have been produced under host-concentrated development environment. In such host-concentrated one, the progress of software development tools has caused several issues. For instance, one of them is that all of software development management has to be suspended when the host computer is down. Since the late 1980s, personal computers have been spread on our daily life instead of conventional mainframe machines, because the price and performance of personal computers have been extremely improved. Hence, computer systems which aid the software development have been also changing into UNIX workstations or personal computers to reduce the development cost. A Client/Server System (CSS) which is a new development method have come into existence as a result of the progress of networking technology by UNIX systems. Such CSS's have been used more and more in the period of network computing. The CSS's are horizontally distributed systems which consist of a server and client computers. The CSS's differ from conventional host/terminal computer systems from the point of view that the CSS's have the property that each computer on network can be a server or client as well. Thus, the CSS's have expanded with the technique of internet. At present, the software development environment has been changing into distributed one because of such progress of network computing technologies. For instance, basic CSS's which consists of 2-layers structure have been expanded to $N$-layers one, because such CSS's can be easily and rapidly introduced for the purpose of software development with low cost. The recent progress of network technologies in social systems is remarkable. As a result of the progress, software development environment has been changing into new development paradigm in such CSS's and distributed development by using network computing technologies (Takahashi, 1998; Umar, 1993; Vaughn, 1994).
The methodology of the object-oriented design and analysis is a feature of such distributed development environment and greatly successful in the field of programming-language, simulation, GUI (graphical user interface), and constructing on database in the software development. A general idea of object-oriented design and analysis is developed as a technique which can easily construct and maintain the complex systems. Therefore, the distributed development paradigm based on such an object-oriented methodology will rapidly grow in the future, because this technique is expected as a very effective approach to

improve software quality and productivity. Software composition by object-oriented technologies is expected as a very effective approach to improve software quality and productivity. Considering the software composition, it is expected that even the host-concentrated development environment can yield the quality of software system to some extent regardless of the content of applications, because the software system is structured on a single hardware environment. On the other hand, it is known that software systems under distributed development environment are difficult to be developed, since the architecture of such systems can have different development styles.

As mentioned above, 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, such Open Source Software (OSS) systems which serve as key components of critical infrastructures in the society are still ever-expanding now (E-Soft Inc.).

Software reliability growth models (SRGM's) (Misra, 1983; Musa et al. 1987; Yamada & Osaki, 1989; Yamada, 1991; Yamada 1994) 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 testing management for the new distributed development paradigm as typified by the open source project has only a few presented (Kuk, 2006; Li et al. 2004; MacCormack et al. 2006; Zhoum & Davis, 2005). 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 paths, size of each component, skill of fault-reporters, and so on.

In this chapter, we discuss a useful reliability assessment method of an embedded OSS developed under open source project. 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 (Karunanithi & Malaiya, 1996; Lippmann, 1987). Also, we propose a software reliability growth model based on stochastic differential equations in order to consider the active state of the open source project. Especially, we apply the intensity of inherent software failures which means the software failure-occurrence rate or the fault detection rate for the $i$ -th component importance level to the interaction among components by introducing an acceleration parameters. Also, we assume that the software failure intensity depends on the time, and the software fault-reporting phenomena on the bug tracking system keep an irregular state in terms of the number of detected faults. Moreover, in order to consider the effect of each software component on the reliability of an entire system under such open source software, we propose a method of reliability assessment based on the Bayesian network (BN) for OSS. Furthermore, we analyze actual software fault-detection count data to show numerical examples of software reliability assessment considering the component importance levels for the open source project.

## 2. Reliability Assessment Method

### 2.1 Weight parameter for each component

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

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

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 $w_{jk}^{2}(j=1,2,\cdots,J;k=1,2,\cdots,K)$ denote the connection weights from $j$-th unit on the association layer to $k$-th unit on the response layer. Moreover, $x_i(i=1,2,\cdots,I)$ represent the normalized input values of $i$-th unit on the sensory layer, and $y_k(k=1,2,\cdots,K)$ are the output values. We apply the normalized values of fault level, operating system, fault repairer, fault reporter to the input values $x_i(i=1,2,\cdots,I)$. Then, 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 $\theta$ 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 (Karunanithi & Malaiya, 1996; Lippmann, 1987). We define the error function by the following equation:

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

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 detected 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 so that the property of the interaction among software components accumulates on the connection weights of neural networks.

By using the output values, $y_k\ (k=1,2,\cdots,K)$, derived from above mentioned method, we can obtain the total weight parameter $p_k$ which represents the level of importance for each component by using the following equation:

$$p_k = \frac{y_k}{\sum_{k=1}^{K} y_k} \quad (k = 1, 2, \ldots, K). \tag{5}$$

### 2.2 Reliability assessment for entire system

Let $S(t)$ be the cumulative number of detected faults in the OSS system by operational time $t\,(t \geq 0)$. Suppose that $S(t)$ takes on continuous real values. Since the 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:

$$\frac{dS(t)}{dt} = \lambda(t) S(t), \tag{6}$$

where $\lambda(t)$ is the intensity of inherent software failures at operational time $t$, and 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-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 software failure intensity depends on the time (Tamura & Yamada, 2007). Therefore, we suppose that $\lambda(t)$ in Eq.(6) has the irregular fluctuation. That is, we extend Eq.(6) to the following stochastic differential equation (Arnold, 1974):

$$\frac{dS(t)}{dt} = \{\lambda(t) + \sigma\gamma(t)\} S(t), \tag{7}$$

where $\sigma$ is a positive constant representing a magnitude of the irregular fluctuation and $\gamma(t)$ a standardized Gaussian white noise. We extend Eq.(7) to the following stochastic differential equation of an Itô type:

$$dS(t) = \left\{ \lambda(t) + \frac{1}{2}\sigma^2 \right\} S(t)dt + \sigma S(t) dW(t), \tag{8}$$

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:

$$\Pr[W(0) = 0] = 1, \tag{9}$$

$$\mathrm{E}[W(t)] = 0, \tag{10}$$

$$\mathrm{E}[W(t)W(t')] = \mathrm{Min}[t,t'], \tag{11}$$

where $\Pr[\mathrm{A}]$ means the probability of event A and $\mathrm{E}[\mathrm{B}]$ represents the expected value of $B$ in the time interval $(0,t]$.

By using Itô's formula (Arnold, 1974), we can obtain the solution of Eq.(7) under the initial condition $S(0) = v$ as follows (Yamada et al. 1994):

$$S(t) = v \cdot \exp\left(\int_0^t \lambda(s)ds + \sigma W(t)\right) \tag{12}$$

where $v$ is the total number of faults detected for the previous software version. Using solution process $S(t)$ in Eq.(12), we can derive several software reliability measures.

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

$$\int_0^t \lambda(s)ds = \sum_{i=1}^{K} p_i\left(1-\exp[-\alpha_i t]\right) \quad \left(\sum_{i=1}^{K} p_i = 1\right), \tag{13}$$

where $\alpha_i (i = 1,2,...,K)$ is an acceleration parameter of the intensity of inherent software failures for the $i$-th component importance level, $p_i\left(\sum_{i=1}^{K} p_i = 1\right)$ the weight parameter for the $i$-th component importance level, and $K$ the number of the applied component. Similarly, we can apply the following S-shaped growth curve to Eq. (12) depending on the trend of fault importance level:

$$\int_0^t \lambda(s)ds = \sum_{i=1}^{K} p_i\left\{1-(1+\alpha_i t)\exp[-\alpha_i t]\right\}. \tag{14}$$

### 2.3 Reliability assessment measures

#### 2.3.1 Expected Number 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

$$f(W(t)) = \frac{1}{\sqrt{2\pi t}}\exp\left\{-\frac{W(t)^2}{2t}\right\}, \tag{15}$$

Data collection on the current total number of detected faults 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. (12) as follows (Yamada et al. 1994):

$$\mathrm{E}\left[S(t)\right] = v \cdot \exp\left( \int_0^t \lambda(s) ds + \frac{\sigma^2}{2} t \right), \tag{16}$$

$$\begin{aligned} \mathrm{Var}\left[S(t)\right] &\equiv \mathrm{E}\left[ \left\{ S(t) - \mathrm{E}\left[S(t)\right] \right\}^2 \right] \\ &= v^2 \cdot \exp\left( 2 \int_0^t \lambda(s) ds + \sigma^2 t \right) \left\{ \exp\left(\sigma^2 t\right) - 1 \right\}, \end{aligned} \tag{17}$$

where $\mathrm{Var}\left[S(t)\right]$ is the variance of the number of faults detected up to time $t$.

### 2.3.2 Mean Time between Software Failures

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

$$MTBF_I(t) = \frac{1}{\mathrm{E}\left[ \frac{dS(t)}{dt} \right]}. \tag{18}$$

Therefore, we have the following instantaneous MTBF:

$$MTBF_I(t) = \frac{1}{v\left( \lambda(t) + \frac{1}{2}\sigma^2 \right) \cdot \exp\left( \int_0^t \lambda(s) ds + \frac{\sigma^2}{2} t \right)}. \tag{19}$$

Also, the cumulative MTBF is approximately given by

$$MTBF_C(t) = \frac{t}{\mathrm{E}\left[S(t)\right]}. \tag{20}$$

Therefore, we have the following cumulative MTBF:

$$MTBF_C(t) = \frac{t}{v \cdot \exp\left( \int_0^t \lambda(s) ds + \frac{\sigma^2}{2} t \right)}. \tag{21}$$

### 2.3.3 Mean Time between Software Failures

Since a one-dimensional Wiener process is a Gaussian process, $\log S(t)$ is a Gaussian process. We can derive its expected value and variance as follows:

$$\mathrm{E}\left[\log S(t)\right] = \log v + \int_0^t \lambda(s)ds, \tag{22}$$

$$\mathrm{Var}\left[\log S(t)\right] = \sigma^2 t. \tag{23}$$

Therefore, we have the following probability for the event $\{\log S(t) \geq x\}$:

$$\Pr\left[\log S(t) \leq x\right] = \Phi\left(\frac{x - \log v - \int_0^t \lambda(s)ds}{\sigma\sqrt{t}}\right), \tag{24}$$

where $\Pr[\mathrm{A}]$ means the probability of event A and $\Phi(\cdot)$ of the standard normal distribution function can defined as follows:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{z^2}{2}\right)dz. \tag{25}$$

Therefore, the transitional probability of $S(t)$ is given by the following equation:

$$\Pr\left[\log S(t) \leq y \middle| S(0) = v\right] = \Phi\left(\frac{\log v + \log y + \int_0^t \lambda(s)ds}{\sigma\sqrt{t}}\right). \tag{26}$$

## 3. Software Reliability Assessment Procedures

The procedures of reliability assessment in 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 fault-detection count data obtained from bug-tracking system, we process the input data for neural network.
3. We estimate the weight parameters $p_i \ (i = 1, 2, \cdots, K)$ for each component by using the neural network.
4. Also, the unknown parameters $\sigma$ and $\alpha_i \ (i = 1, 2, \cdots, K)$ included in our model are estimated by using the least-square method of Marquardt-Levenberg.
5. We show the expected total number of detected faults, the instantaneous fault-detection rate, and the cumulative MTBF as software reliability assessment measures, and the predicted relative error.

## 4. Portability Assessment

### 4.1 Prior information for BN

Applying SRGM's for prior information in case of using BN, we analyze software fault-detection count data based on an NHPP model. Considering stochastic characteristics associated with the fault-detection procedures in the testing phase, we treat $\{N(t),\ t \geq 0\}$ as a nonnegative counting process where a random variable $N(t)$ means the cumulative number of faults detected up to testing-time $t$. The fault-detection process $\{N(t),\ t \geq 0\}$ are described as follows:

$$\Pr\left[N(t)=n\right]=\frac{\left\{H(t)\right\}^{n}}{n!}\exp\left[-H(t)\right] \quad (n=0,1,2,...) \tag{27}$$

In Eq. (27), $\Pr[\mathrm{A}]$ 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 testing-time interval $(0,\ t]$, where $h(t)$ is called an intensity function which represents the fault-detection rate per one fault. From Eq. (27), the fault-detection rate per one remaining fault, which characterizes the software reliability growth in the fault-detection phenomenon, is defined as

$$d(t) \equiv \frac{h(t)}{a-H(t)}, \tag{28}$$

where $a$ is the expected number of the initial inherent faults. We can analyze software fault-detection count data by using an SRGM based on the NHPP, because the NHPP models have been discussed in many literatures since they can be easily applied in the software reliability assessment. The SRGM based on an NHPP is based on the following assumptions :

- A software fault-detection phenomenon can be described by an NHPP.
- Software faults detected during the testing-phase are corrected certainly and completely, i.e., no new faults are introduced into the software system during the debugging.

It is empirically known that the cumulative number of detected faults shows an exponential growth curve when a software system consisting of several software components are tested in the testing-phase. On the other hand, the cumulative number of faults describes an S-shaped growth curve when a newly developed software system is tested. Thus, the former case is described by the exponential SRGM based on an NHPP, and the latter case is described by the delayed S-shaped SRGM which is also based on an NHPP. We describe the structure of the mean value function defined in the following, because an NHPP model is characterized by its mean value function. The mean value function of the exponential SRGM, $H_e(t)$, is characterized by the following function:

$$H_e(t)=a\left(1-e^{-bt}\right) \qquad (a>0,\ b>0), \tag{29}$$

where $H_e(t)$ represents the expected cumulative number of faults detected up to the module testing time $t$ $(t \geq 0)$. In Eq. (29), $a$ is the expected number of initial inherent faults, and $b$ the software failure rate per inherent fault. In addition, the intensity function of the exponential SRGM is given as follows:

$$h_e(t) = abe^{-bt}, \tag{30}$$

where $h_e(t)$ represents the instantaneous fault-detection rate at the module testing time $t$ $(t \geq 0)$.

Similarly, the mean value function of the delayed S-shaped SRGM, $H_s(t)$, is represented as :

$$H_s(t) = a\left[1-(1+bt)e^{-bt}\right] \qquad (a>0,\ b>0), \tag{31}$$

where $H_s(t)$ represents the expected cumulative number of faults detected up to the module testing time $t$ $(t \geq 0)$. In Eq. (6), $a$ is the expected number of initial inherent faults, and $b$ the software failure rate per inherent fault. In addition, the intensity function of the delayed S-shaped SRGM is given as follows :

$$h_s(t) = ab^2te^{-bt} \tag{32}$$

where $h_s(t)$ represents the instantaneous fault-detection rate at the module testing time $t$ $(t \geq 0)$.

In this chapter, we use Mean Squared Errors (MSE) discussed in 5.3 in order to select a better SRGM, Exponential SRGM or Delayed S-shaped SRGM for each software component.

### 4.2 Reliability assessment for OSS porting

We can estimate the probability of the phenomenon $Z$ based on the following phenomena $X$ and $Y$:

$X$: The fault is detected at the component $X$.
$Y$: The fault is detected at the component $Y$.
$Z$: The fault is detected at the Kernel component.

BN for above mentioned phenomena is shown in Fig. 1.

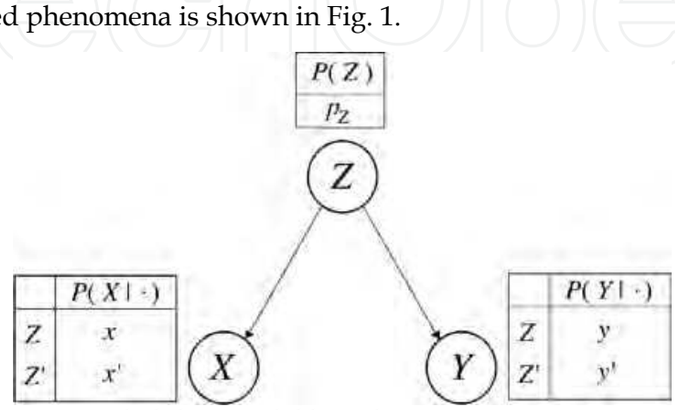



Fig. 1. The failure-occurrence probability model based on BN.

Fig. 1 means that the fault is detected at the component $X$ or component $Y$ as a result of the occurrence of phenomenon $Z$. Therefore, the failure probability of Kernel component is given by the following equation based on the Bayesian theory:

$$p_Z = \frac{xyp'_Z b_x b_y}{xyp'_Z + x'y'\overline{p}'_Z} + \frac{\overline{x}yp'_Z \overline{b}_x b_y}{\overline{x}yp'_Z + \overline{x}'y'\overline{p}'_Z} + \frac{x\overline{y}p'_Z b_x \overline{b}_y}{x\overline{y}p'_Z + x'\overline{y}'\overline{p}'_Z} + \frac{\overline{x}\overline{y}p'_Z \overline{b}_x \overline{b}_y}{\overline{x}\overline{y}p'_Z + \overline{x}'\overline{y}'\overline{p}'_Z}, \tag{33}$$

where $x$ is the probability of $X$ under the occurrence of phenomenon $Z$. Similarly, $y$ is the probability of $Y$ under the occurrence of phenomenon $Z$. The probability of $X$ and $Y$ are given by the $b_x$ and $b_y$ previously. Also, $\overline{p}$ means the $1-p$. $p'_Z$ is the prior probability of $Z$, $p'_Z$ is updated by using $p'_Z = p_Z$. The prior information $b_x$ and $b_y$ are given by the intensity function of NHPP model and SDE model applied for each component of OSS. We can estimate the portability of embedded software based on OSS by using Eq. (33).

## 5. Numerical Examples

### 5.1 Embedded OSS

We focus on the BusyBox (Erik Andersen) which is one of the embedded open source software developed under an open source project. The BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc.

The fault-detection count data used in this chapter are collected in the bug tracking system on the website of BUSYBOX in August 2008.

### 5.2 Reliability assessment considering component level

Estimating the weight parameters based on the number of source lines of code for each component, we analyze the actual data for the case of $v = 14,\ p_1 = 0.28252,\ p_2 = 0.07213,\ \hat{\alpha}_3 = 0.64531.$

In case of the exponential intensity function of inherent software failures in Eq. (30), the following model parameters have been estimated:

$$\hat{\alpha}_1 = 0.324667,\ \hat{\alpha}_2 = 0.116271,\ \hat{\alpha}_3 = 0.121832,\ \hat{\sigma} = 0.153831.$$

In case of the S-shaped intensity function of inherent software failures in Eq. (32), the following model parameters have been estimated:

$$\hat{\alpha}_1 = 0.141554,\ \hat{\alpha}_2 = 0.168274,\ \hat{\alpha}_3 = 0.567777,\ \hat{\sigma} = 0.153845.$$

The estimated expected number of detected faults, $\hat{\mathrm{E}}[S(t)]$, in case of the exponential intensity function of inherent software failures, is shown in Fig. 2. Also, Fig. 3 shows the

estimated variance of the number of faults. From Fig. 3, we find that the variance of the number of detected faults grows as the time elapses after the release.

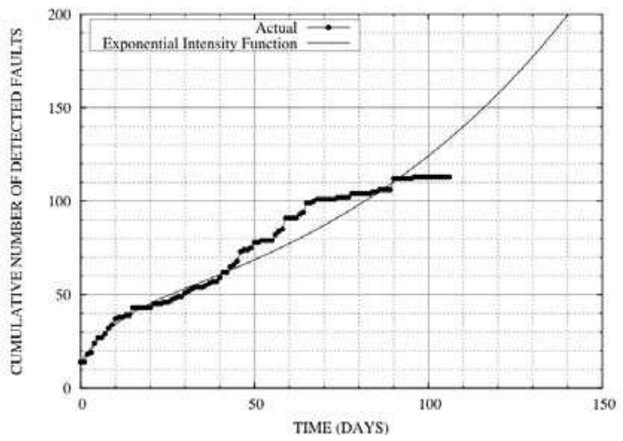



Fig. 2. The estimated expected cumulative number of detected faults.

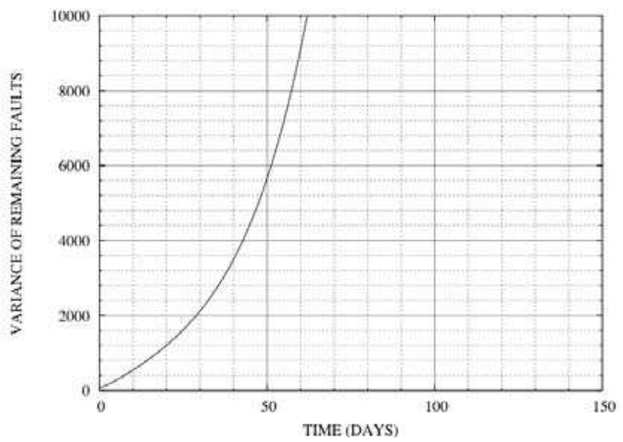



Fig. 3. The estimated variance of the number of detected faults.

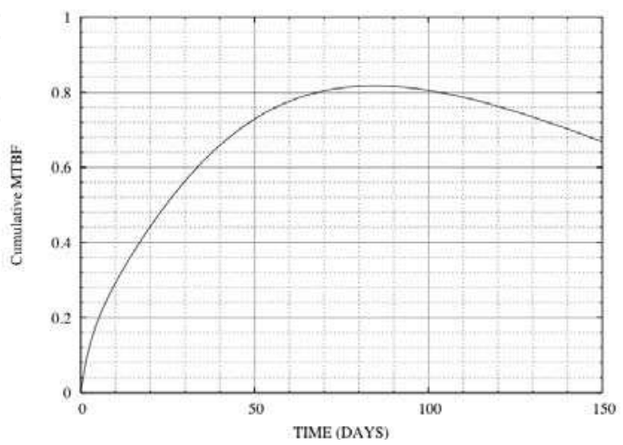



Fig. 4. The estimated $MTBF_C$.

Moreover, the estimated $MTBF_C$ is also plotted in Fig. 4. Fig. 4 shows that the MTBF increase as the operational procedures go on.
Furthermore, Fig. 5 shows the estimated transitional probability of Eq. (26) in case of the exponential intensity function of inherent software failures.

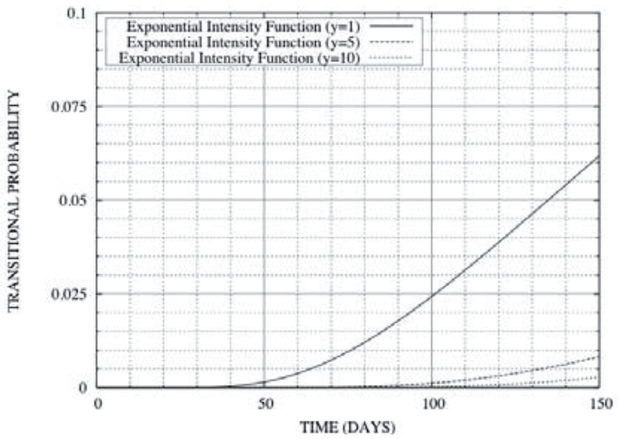



Fig. 5. The estimated transitional probability of $S(t)$.

### 5.3 Assessment Measures for OSS porting

We show numerical example in terms of OSS porting by using the fault-detection count data registered in the bug tracking system in actual open source project. In this chapter, we apply the fault data of BusyBox that is developed and used as Embedded OSS.
We focus on the buildroot and the uClibc which is one of the components in the Busybox. We apply the mean value functions of exponential SRGM and delayed S-shaped SRGM for each components. We show the result of parameter estimation and Mean Square Error (MSE) in Tables 1-3. Moreover, we show the estimation of expected values of the cumulative number of detected faults in Figs. 6-8. We define the error function in Eq. (34) by the following equation :

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \hat{y}_n\right)^2, \tag{34}$$

where $y_k$ are the actual measurement values, and $\hat{y}_k$ are the predictive values. The MSE indicates that the selected model fits better to the observed data as MSE becomes small.

| | a | b | MSE |
|---|---|---|---|
| Exponential SRGM | 155.82 | 0.0026 | 5.3916 |
| Delayed S-shaped SRGM | 43.966 | 0.0305 | 2.2497 |

Table 1. The results of estimated unknown parameter and value of MSE for each SRGM in BusyBox.

| | a | b | MSE |
|---|---|---|---|
| Exponential SRGM | 159.19 | 0.0039 | 4.3498 |
| Delayed S-shaped SRGM | 60.7672 | 0.0318 | 5.7175 |

Table 2. The results of estimated unknown parameter and value of MSE for each SRGM in buildroot.

| | a | b | MSE |
|---|---|---|---|
| Exponential SRGM | 28.270 | 0.0056 | 0.8490 |
| Delayed S-shaped SRGM | 13.758 | 0.0351 | 0.3729 |

Table 3. The results of estimated unknown parameter and value of MSE for each SRGM in uClibc.

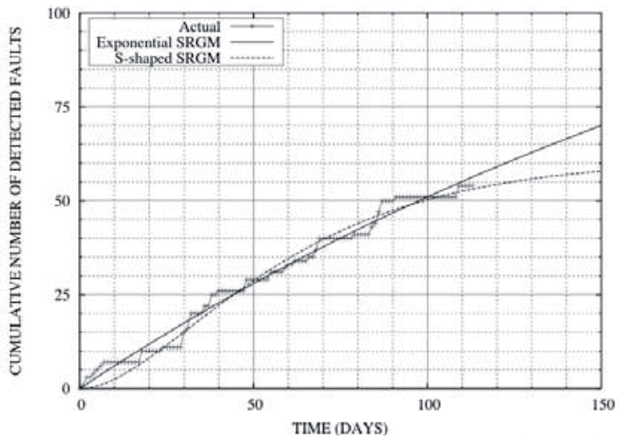



Fig. 6. The estimated expected cumulative number of detected faults for buildroot.

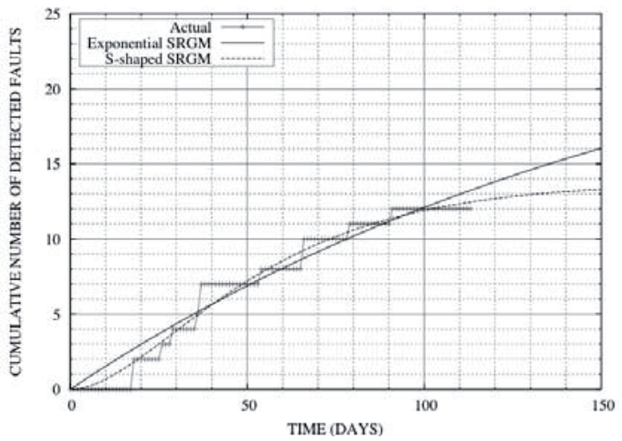



Fig. 7. The estimated expected cumulative number of detected faults for uClibc.

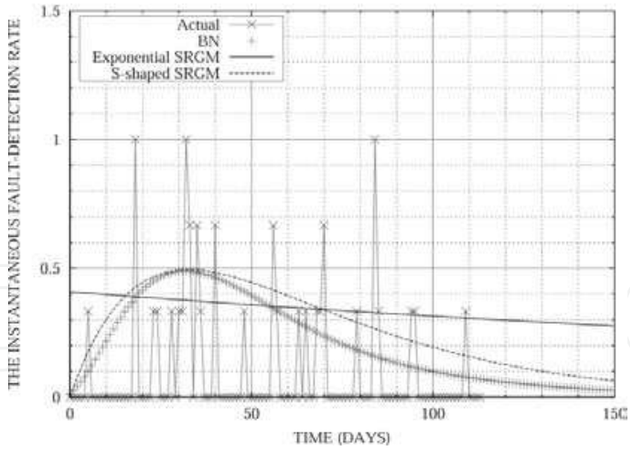



Fig. 8. The instantaneous fault-detection rate by using BN.

We can apply the exponential SRGM for buildroot from Fig. 6 and Table 2. On the other hand, we can apply the delayed S-shaped SRGM for uClibc from Fig. 7 and Table 3. Fig. 8 is shown the instantaneous fault-detection rate by using BN. From Fig. 8, we find that the fault-detection rate can be used as the portability assessment measure for the embedded OSS by using BN.

## 6. Conclusion

In this chapter, we have discussed the methods of reliability analysis for an embedded OSS developed under on an open source project. Especially, we have proposed a stochastic differential equation model in order to consider the active state of the open source project, where 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. Moreover, we have derived several assessment measures in terms of imperfect debugging of an entire system under such open source development paradigm. Especially, we have applied the exponential and S-shaped intensity functions of inherent software failures for the $i$-th component importance level to the interaction among components by using an acceleration parameters. Furthermore, we have shown the transitional probability of $S(t)$ in Eq. (26).

In case of considering the effect of debugging process on an entire system on software reliability assessment for open source projects, it is necessary to grasp the deeply-intertwined factors. In this chapter, we have shown that our method can describe such deeply-intertwined factors. Especially, we have applied BN technique 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 BN, we have proposed the method of reliability assessment incorporating the interaction among software components. Moreover, we have proposed the fault-detection rate based on BN used as the portability assessment measure for the embedded OSS.

Finally, we have focused on an embedded 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. Acknowledgments

This work was supported in part by the Grant-in-Aid for Young Scientists (B), Grant No. 21700044 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

## 8. References

Arnold, L. (1974) *Stochastic Differential Equations-Theory and Applications*, John Wiley & Sons, New York

Erik Andersen, BUSYBOX. [Online]. Available: http://www.busybox.net/E-Soft Inc., Internet Research Reports. [Online].
Available: http://www.securityspace.com/s_survey/data/

Kuk, G. (2006) Strategic interaction and knowledge sharing in the KDE developer mailing list, *Informs J. Management Science*, Vol. 52, No. 7, pp. 1031-1042

Karunanithi, N. & Malaiya, Y. K. (1996) Neural networks for software reliability engineering, *Handbook of Software Reliability Engineering* M. R. Lyu (ed.), pp. 699-728, McGraw-Hill, New York

Li, P.; Shaw, M.; Herbsleb, J.; Ray, B. & Santhanam, P. (2004). 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)*, pp. 263-272

Lippmann, R. P. (1987) An introduction to computing with neural networks, *IEEE Trans. ASSP*, Vol. 4, No. 2, pp. 4-22

MacCormack, A.; Rusnak, J. & Baldwin, C.Y. (2006) Exploring the structure of complex software designs: an empirical study of open source and proprietary code, *Informs J. Management Science*, Vol. 52, No. 7, pp. 1015-1030

Misra, P. N. (1983) Software reliability analysis, *IBM Systems J.*, Vol. 22, No. 3, pp. 262-270

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

Takahashi, M. (1998) *The Method of Effort Estimation under Client/Server System Development: Models and Applications (in Japanese)*, Soft Research Center, Tokyo

Tamura, Y. & Yamada, S. (2007) Software reliability assessment and optimal version-upgrade problem for open source software, *Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1333-1338 Montreal, Canada

Tamura, Y. & Yamada, S. (2008) A method of reliability assessment based on deterministic chaos theory for an open source software, *Proceedings of the Second IEEE International Conference on Secure System Integration and Reliability Improvement*, pp. 60-66, Yokohama, Japan

Tamura, Y. & Yamada, S. (2008) Comparison of software reliability assessment methods based on deterministic chaos theory for an open source software, *Proceedings of the 2008 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3606-3611, Suntec, Singapore

Umar, A. (1993) *Distributed Computing and Client-Server Systems*, Prentice Hall, Englewood Cliffs, New Jersey

Vaughn, L. T. (1994) *Client/Server System Design and Implementation*, McGraw-Hill, New York

Yamada, S. & Osaki, S. (1985) Software reliability growth modeling: Models and applications, *IEEE Trans. Software Engineering*, Vol. SE-11, No. 12, pp. 1431-1437

Yamada, S. (1991) Software quality/reliability measurement and assessment: Software reliability growth models and data analysis, *J. information Processing*, Vol. 14, No. 3, pp. 254-266

Yamada, S. (1994). *Software Reliability Models: Fundamentals and Applications (in Japanese)*, JUSE Press, Tokyo

Yamada, S.; Kimura, M.; Tanaka, H. & Osaki, S. (1994) Software reliability measurement and assessment with stochastic differential equations, *IEICE Trans. Fundamentals*, Vol. E77-A, No. 1, pp. 109-116

Zhoum, Y. & Davis, J. (2005) Open source software reliability model: an empirical approach, *Proceedings of the Workshop on Open Source Software Engineering (WOSSE)*, pp.67-72, Vol. 30, No. 4

IntechOpen




**Mechatronic Systems Simulation Modeling and Control**

Edited by Annalisa Milella Donato Di Paola and Grazia Cicirelli

ISBN 978-953-307-041-4
Hard cover, 298 pages
**Publisher** InTech
**Published online** 01, March, 2010
**Published in print edition** March, 2010

This book collects fifteen relevant papers in the field of mechatronic systems. Mechatronics, the synergistic blend of mechanics, electronics, and computer science, integrates the best design practices with the most advanced technologies to realize high-quality products, guaranteeing at the same time a substantial reduction in development time and cost. Topics covered in this book include simulation, modelling and control of electromechanical machines, machine components, and mechatronic vehicles. New software tools, integrated development environments, and systematic design methods are also introduced. The editors are extremely grateful to all the authors for their valuable contributions. The book begins with eight chapters related to modelling and control of electromechanical machines and machine components. Chapter 9 presents a nonlinear model for the control of a three-DOF helicopter. A helicopter model and a control method of the model are also presented and validated experimentally in Chapter 10. Chapter 11 introduces a planar laboratory testbed for the simulation of autonomous proximity manoeuvres of a uniquely control actuator configured spacecraft. Integrated methods of simulation and Real-Time control aiming at improving the efficiency of an iterative design process of control systems are presented in Chapter 12. Reliability analysis methods for an embedded Open Source Software (OSS) are discussed in Chapter 13. A new specification technique for the conceptual design of self-optimizing mechatronic systems is presented in Chapter 14. Chapter 15 provides a general overview of design specificities including mechanical and control considerations for micro-mechatronic structures. It also presents an example of a new optimal synthesis method to design topology and associated robust control methodologies for monolithic compliant microstructures.

**How to reference**

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Yoshinobu Tamura and Shigeru Yamada (2010). Reliability Analysis Methods for an Embedded Open Source Software, Mechatronic Systems Simulation Modeling and Control, Annalisa Milella Donato Di Paola and Grazia Cicirelli (Ed.), ISBN: 978-953-307-041-4, InTech, Available from: http://www.intechopen.com/books/mechatronic-systems-simulation-modeling-and-control/reliability-analysis-methods-for-an-embedded-open-source-software

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai

Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen