Reliability Modeling Incorporating Error Processes for Internet-Distributed Software

Jung-Hua Lo, Sy-Yen Kuo, *Fellow, IEEE* and Chin-Yu Huang

***Abstract*—This paper proposes several improvements on the conventional software reliability growth models (SRGMs) to describe actual software development process by eliminating an unrealistic assumption that detected errors are immediately corrected. A key part of the proposed models is the "delay-effect factor", which measures the expected time lag in correcting the detected faults during software development. To establish the proposed model, we first determine the delay-effect factor to be included in the actual correction process. For the conventional SRGMs, the delay-effect factor is basically non-decreasing. This means that the delayed effect becomes more significant as time moves forward. Since this phenomenon may not be reasonable for some applications, we adopt a bell-shaped curve to reflect human learning process in our proposed model. Experiments on a real data set for internet-distributed software has been performed, and the results show that the proposed new model gives better performance in estimating the number of initial faults than previous approaches.**

***Index Terms*— Delay-Effect Factor, Fault Detection/Correction Processes, Delayed-Time NHPP Model, SRGM.**

I. INTRODUCTION

Due to the recent rapid developments of computer and network technologies, the Internet and World Wide Web make it possible for users to access a variety of resources and applications distributed over the world. With its huge popularity, the Internet has a very important feature that a large body of software is freely available and widely distributed on it, and may be obtained from various accessible archives [1-2]. This software includes shareware, free packages and libraries, and other software programs that the designers contribute for free. Therefore, a software failure on the Internet could result in big loss or degradation of service to the sharing components among the software products.

Moreover, there are some similarities between the Internet software and traditional software. Firstly, they are extensively reused. Examples of the former are Java applets, ActiveX controls, software agents, Javascripts, VBscripts, etc [3,4]. Comparatively, traditional software packages also have reuse property; e.g., reuse of subroutines and macros [2]. Furthermore, both hold the primitive nature of software, such as the original causes resulting in system unreliability, copies from a software program are identical, the basic strategies of dealing with fault tolerance, and other inherent properties: no aging, no wear-out, etc. Therefore, conventional repair and maintenance policies can be used for the Internet software [2-5].

Since software is embedded in network technologies and permeates our daily life, the correct performance of internet-distributed software systems becomes an important issue of many critical systems. Software reliability can be viewed as a powerful measure of quantifying software failures and it is defined as the probability of failure-free software operation for a specified period of time in a specified environment [5]. Therefore, in order to achieve desired level of quality, the software reliability of a software system must be carefully evaluated.

The fault-detection and fault-correction are critical processes in attaining good performance of software quality. During the software detection process, testing cases are run and ultimately failures are detected. After detection, the debugging team should analyze the failure, locate the fault, and fix the fault [6-8]. Numerous stochastic models for the software failure scenario have been developed to measure software reliability, and some of them are based on the Nonhomogeneous Poisson Process (NHPP). In practice, these software reliability growth models (SRGMs) are very useful to describe the error-detection process as a discrete or continuous process with a time-dependent error-detection rate [5-8]. The general assumption in conventional SRGMs is that the detected faults are immediately removed. From a practical point of view, this assumption is not valid in a real software environment [5-8,16]. We know that the time to remove a fault depends on the complexity of the detected errors, the skills of the debugging team, the available manpower, and the software development environment [8, 17-18]. This phenomenon can especially be seen when some latent software errors are hard to detect and they even exist in the software product for a long time after they are detected. Therefore, the time delayed by the correction process is not negligible. The objective here is to remove this assumption in order to make the SRGMs more realistic and applicable. Schneidewind [6, 16] first modeled the fault-correction process by using a constant-delay fault-detection process. Xie [6] extended the Schneidewind model to a continuous version by substituting a time-dependent delay function for the constant delay. A key factor of the continuous version of Schneidewind model is the time-dependent delay function, which measures the expected

This research was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 89-2213-E002-114.

Jung-Hua Lo and Sy-Yen Kuo are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (e-mail: sykuo@cc.ee.ntu.edu.tw).

Chin-Yu Huang is with the Central Bank of China, Taipei, Taiwan (e-mail: cy2huang@mail.cbc.gov.tw).

IEEE Catalogue No. 01CH37239
0-7803-7101-1/ 01 / $10.00 © 2001 IEEE.

time lag to correct a detected fault. Furthermore, we observe that the time-dependent delay of the conventional SRGMs has three possible trends as the processes proceed: constant, bounded increasing, and unbounded increasing. That is, the time delayed by the correction activities is increasing as time progresses and these three trends are not always reasonable in many applications. We thus propose a new time-dependent delay function based on the bell-shaped curve to represent the human learning process.

In particular, if we focus on the Internet-distributed software packages, the users will send messages to report bugs via the Internet. Occasionally, some users may misuse the software and send inapplicable messages; other users may report the same problem, which leads to duplication [1-2]. Therefore, the debugging team should spend their time to distinguish between user-error and actual faults. That is, the correction process in the Internet-distributed software has a more important role than in the traditional software applications. Our experimental data will demonstrate this peculiar phenomenon and the results show that the proposed models give better performance in estimating the number of initial faults.

In this paper, we also show how several existing SRGMs [8-11, 14-15,19] based on NHPP models can be completely derived by applying the time-dependent delay function. Furthermore, we observe that the time-dependent delay of the mentioned models has three possible trends as the processes proceed: constant, bounded increasing, and unbounded increasing. That is, the time delayed by the correction activities is increasing as time progresses. Therefore, these three trends are not always reasonable in many applications. We thus propose a new time-dependent delay function based on the bell-shaped curve to represent the human learning process. Experiments have been performed based on two real data sets, and the results show that the proposed models give better performance in estimating the number of initial faults and also indicate a goodness-of-fit in terms of the mean of squares errors criterion. Besides, it was observed that the proposed model is good at predicting the behavior of future failures. On the other hand, we discuss the relationship between delayed-time and non delayed-time NHPP models, and derive a general scheme of delayed-time NHPP models from a new viewpoint [22].

There are six sections in this paper. Section 2 gives a description of characteristics of the NHPP models with delayed correction process and shows how some existing NHPP models can be reinterpreted from a viewpoint of delayed correction process. Several new NHPP models are proposed in Section 3. Section 4 provides a general scheme of delayed-time NHPP models. The experiments and results are presented in Section 5. Finally, Section 6 concludes this paper.

II. MOTIVATIONS AND BACKGROUNDS

Most existing models under the assumption that the detected faults are immediately corrected can be regarded as delayed fault-detection models that can model the fault-detection and fault-correction processes. That is, we can remove the impractical assumption that the fault-correction process is perfect and can thus establish a corresponding time-dependent delay function to fit the fault-correction process. Thus, we propose a new model, using the new assumptions:

Definition 1 (Delay-effect factor): Given a fault-detection and fault-correction process, one defines the delay-effect factor, $\varphi(t)$, to be a time-dependent function that measures the expected delay in correcting a detected fault at any time.

Definition 2 (Delayed-time NHPP model): A software reliability model is called a delayed-time NHPP model if it obeys the following assumptions: [5-8,16]

1. The error-detection process follows the NHPP and all faults are independent and equally detectable.
2. The software system is subject to exploring the remaining faults in the system at random times.
3. The rate of change of the mean value function is exponentially decreasing.
4. The detected error is not immediately removed and lags by a delay-effect factor, $\varphi(t)$.

Based on the above assumptions 1-3, the original mean value function of NHPP model is

$$m_{original}(t) = a(1-e^{-bt}), \quad a>0, \quad b>0 \qquad (1)$$

where a = the expected number of initial faults, and b = the fault-detection rate.

From assumption 4 in Definition 2 and Eq. (1), the new mean value function is

$$m(t) = m_{original}(t-\varphi(t)) = a(1-e^{-bt}e^{b\varphi(t)}). \qquad (2)$$

Theorem 1: Given a delay-effect factor, $\varphi(t)$, we have

(a) The fault-detection intensity of the delayed-time NHPP model is

$$\lambda(t) = abe^{-bt}e^{b\varphi(t)}(1-\tfrac{d\varphi(t)}{dt}), \quad a>0, \quad b>0. \qquad (3)$$

(b) $d\varphi(t)/dt < 1$

(c) The probability that no errors occur in the interval $(s, s+t)$ given time s is

$$R(t|s) = \exp[-a(e^{-b(t+s)}e^{b\varphi(t+s)} - e^{-bs}e^{b\varphi(s)})]. \qquad (4)$$

□

As an example, if we assume that the fault-detection intensity is $\lambda(t) = ab^2e^{-bt}$, we can get its corresponding mean value function $m(t)$ by the integration of $\lambda(t)$ as shown below:

$$m(t) = \int \lambda(t)dt = a(1-(1+bt)e^{-bt}), \quad a>0, \quad b>0. \qquad (5)$$

That is, a variation of the G-O model, known as the delayed S-shape model [8], can be derived. On the other hand, if we assume the delay-effect factor is $\varphi(t) = \frac{1}{b}\ln(1+bt)$, we can also get its corresponding mean value function by Eq. (2) as below:

$m(t) = a(1-e^{-bt}e^{b\varphi(t)}) = a(1-(1+bt)e^{-bt}),\ \ a>0,\ \ b>0.$

This example reflects the fact the S-shape model can be interpreted from various points of view. In other words, by specifying the fault-detection rate or the delay-effect factor, we can formulate various models with a new mean value function of the NHPP.

In the following, we show that several classical SRGMs [7-10, 14, 15, 17] that are based on NHPP can be directly derived from Definition 1, Definition 2 and Theorem 1. Furthermore, we also derive the fault-detection intensity from Eq. (3) and check the condition of Theorem 1 (i.e. $\frac{d\varphi(t)}{dt} < 1$):

- **Goel-Okumoto model [9]**

Take $\varphi(t) = 0$, then we have

(a) $d\varphi(t)/dt = 0 < 1,$

(b) the mean value function (MVF),

$m(t) = a(1-e^{-bt}),\ \ a>0,\ \ b>0,$

(c) the fault-detection intensity (FDI),

$\lambda(t) = abe^{-bt},\ \ a>0,\ \ b>0.$

- **Gompertz growth curve model [9]**

Take $\varphi(t) = t + \frac{1}{b}\ln(1-ke^{-bt})$, then we have

(a) $d\varphi(t)/dt = 1+\ln k(k^{e^{-bt}}e^{-bt}/1-k^{e^{-bt}}) < 1,$

(b) MVF, $m(t) = ak^{e^{bt}},\ \ a>0,\ \ b>0,\ \ 1>k>0,$

(c) FDI, $\lambda(t) = ab(\ln k)e^{bt}k^{e^{bt}},\ \ a>0,\ \ b>0,\ \ 1>k>0.$

- **Logistic growth curve model [11-13]**

Take $\varphi(t) = \frac{1}{b}\ln\frac{k}{1+ke^{-bt}}$, then we have

(a) $d\varphi(t)/dt = (ke^{-bt}/1+ke^{-bt}) < 1,$

(b) MVF, $m(t) = a\frac{1}{1+ke^{-bt}},\ \ a>0,\ \ b>0,\ \ k>0,$

(c) FDI, $\lambda(t) = \frac{abke^{-bt}}{(1+ke^{-bt})^2},\ \ a>0,\ \ b>0,\ \ k>0.$

- **Log-logistic growth curve model [7]**

Take $\varphi(t) = \frac{1}{b}\ln(\frac{e^{bt}}{1+\lambda^k e^{k\ln t}})$, then we have

(a) $d\varphi(t)/dt = 1-\frac{k\lambda^k e^{k\ln t}}{bt(1+\lambda^k e^{k\ln t})} < 1,$

(b) MVF, $m(t) = a\frac{(\lambda t)^k}{1+(\lambda t)^k},\ \ a>0,\ \ \lambda>0,\ \ k>0,$

(c) FDI, $\lambda(t) = \frac{ak\lambda^k t^{k-1}}{(1+(\lambda t)^k)^2},\ \ a>0,\ \ \lambda>0,\ \ k>0.$

- **Delayed S-shaped model [8]**

Take $\varphi(t) = \frac{1}{b}\ln(1+bt)$, then we have

(a) $d\varphi(t)/dt = \frac{1}{1+bt} < 1,$

(b) MVF, $m(t) = a[1-(1+bt)e^{-bt}],\ \ a>0,\ \ b>0,$

(c) FDI, $\lambda(t) = ab^2te^{-bt},\ \ a>0,\ \ b>0.$

- **Inflected S-shaped model [8]**

Take $\varphi(t) = \frac{1}{b}\ln(\frac{c+1}{1+ce^{-bt}})$, then we have

(a) $d\varphi(t)/dt = \frac{ce^{-bt}}{(c+1)(1+ce^{-bt})} < 1,$

(b) MVF, $m(t) = a\frac{(1-e^{-bt})}{(1+ce^{-bt})},\ \ a>0,\ \ b>0,\ \ c>0,$

(c) FDI, $\lambda(t) = \frac{abce^{-bt}}{(1+ce^{-bt})^2},\ \ a>0,\ \ b>0,\ \ c>0.$

- **Modified Duane model [14]**

Take $\varphi(t) = \frac{\gamma}{b}\ln(\frac{\beta}{\beta+t}) + t$, then we have

(a) $d\varphi(t)/dt = 1-\frac{\beta}{b(\alpha+t)} < 1,$

(b) MVF, $m(t) = a\{1-(\frac{\beta}{\beta+t})^\gamma\},\ \ a>0,\ \ \beta>0,\ \ \gamma>0,$

(c) FDI, $\lambda(t) = \frac{a\gamma\beta^\gamma}{(\beta+t)^{\gamma+1}},\ \ a>0,\ \ \beta>0,\ \ \gamma>0,$

- **Weibull-type testing-effort function model [15]**

Take $\varphi(t) = t + \alpha\exp(-\beta t^\gamma) - \alpha$, then we have

(a) $d\varphi(t)/dt = 1-\alpha\beta\gamma t^{\gamma-1}\exp(-\beta t^\gamma) < 1,$

(b) MVF,

$m(t) = a\{1-\exp[-b\alpha(1-e^{-\beta t^\gamma})]\}, a, b, \alpha, \beta, \gamma > 0,$

(c) FDI,

$\lambda(t) = ab\alpha\beta\gamma t^{\gamma-1}e^{-\beta t^\gamma}\exp[-b\alpha(1-e^{-\beta t^\gamma})]\}, a, b, \alpha, \beta, \gamma > 0.$

From the above derivation, we find that the delay-effect factor can be one of the following three types:

Case 1: (*Constant delay-effect factor*)

This is the trivial case that the delay-effect factor is a constant and thus has equal time lag through all the fault-detection processes. For example, when the delay-effect factor is a nonzero constant value (Λ), we have the following mean value function:

$m(t) = a(1-ke^{-bt}),\ \ a>0,\ \ b>0,\ \ 1>k>0,$

where a= the expected number of initial faults,
b= the fault-detection rate, and
$k = exp(-b\Lambda)$.

This means that it has $a\times(1-k)$ errors detected (i.e. $m(0)=a\times(1-k)$) at the beginning of the software testing process. Apparently, the G-O model belongs to this type and the original G-O model is obtained when the constant is zero.

Case 2: (*Unbounded increasing delay-effect factor*)
In this case, the delay-effect factor is increasing as the software test process proceeds and approaches an unlimited value as time reaches infinity. This means that the delayed-time phenomenon is more serious as testing progresses, and the time lag is infinite as time approaches infinity. This phenomenon, however, contradicts the human learning process. The delay-effect factors of this kind in Section 2 are listed as follows:

- $t+\frac{1}{b}\ln(1-ke^{-bt})$
- $\frac{1}{b}\ln(1+bt)$
- $\frac{\gamma}{b}\ln(\frac{\beta}{\beta+t})+t$
- $t+\alpha\exp(-\beta t^{\gamma})-\alpha$
- $\frac{1}{b}\ln(\frac{e^{bt}}{1+\lambda^{k}e^{k\ln t}})$

Case 3: (*Bounded increasing delay-effect factor*)
The delay-effect factor is lower at the beginning of the software testing and increases as testing proceeds. Unlike the unbounded increasing case, the time lag is restricted to a constant value beyond which it cannot increase. In other words, the time-lag effect is under control. This may be due to the deadline for the delivery of the software product, for example. The delay-effect factors of this kind in section 2 are listed as follows:

- $\frac{1}{b}\ln(\frac{c+1}{1+ce^{-bt}})$

Intuitively, the correction process can be viewed as a learning process that the software testing teams become familiar with the debugging environments and tools as time progresses, and these teams' skills gradually improve and thus the amount of time lag will be smaller. In other words, the delay-effect factor is non-increasing under the above circumstances.

III. INCORPORATING BELL-SHAPED DELAY-EFFECT FACTOR INTO SRGMS

In Section 2 we reconstructed some conventional NHPP models based on the delay-effect factor instead of the unrealistic assumption that the detected errors are instantly corrected. Moreover, from the derived NHPP models we conclude that the above delay-effect factors are non-decreasing (i.e. constant, unbounded increasing or bounded increasing). In reality, the delay-effect factor is a function of the complexity of program modules, the manpower, the skill of testing teams, the deadline for the release of the software, etc. At the beginning of the software detection and correction process, the programmers could remove easy-to-detect errors in their own programs and thus the extent of delayed effect is limited. As time goes, it is relatively more difficult for the debugging team to correct more errors. That is, the delay-effect factor is increasing in this testing phase. As time passes further, the debugging team becomes acquainted with the software-testing environment, with better skills, techniques and tools. These improvements may speed up the testing activities [5,11-13,17-18]. That is, the time lag that resulted from the correction process becomes decreasing. In order to incorporate the above inference and make the fault-detection and fault-correction models more practical, we adopt the bell-shaped delay-effect factor, which reflects the human learning ability, to fit the increasing-then-decreasing scenario of the delayed fault-detection process. In addition, we introduce the bell-shaped functions, i.e., **Gamma curve, Rayleigh curve** and **Weibull curve**, to describe the testing/debugging activities. For a Gamma curve, we often denote it by

$$GAM(\theta,\kappa)=cx^{\kappa-1}e^{-\frac{x}{\theta}},\quad c>0,\ \kappa>0,\ \theta>0$$

where the parameter κ is called the **shape parameter** because it determines the basic shape of the graph of the curve. In particular, there are three basic shapes, depending on whether $\kappa<1$, $\kappa=1$ and $\kappa>1$. In this paper, we focus on the bell-shaped curve with shape parameter $\kappa=2$, 3 and 4. Similarly, a Weibull curve has the following notation:

$$WEI(\theta,\kappa)=cx^{\kappa-1}e^{-(\frac{x}{\theta})^{\kappa}},\quad c>0,\ \kappa>0,\ \theta>0$$

where the parameter κ is also called a **shape parameter** and has three basic shapes, depending on whether $\kappa<1$, $\kappa=1$ and $\kappa>1$. The special case with $\kappa=2$ is known as the Rayleigh curve. Note that when $\kappa \le 1$ the curve is not bell-shaped: it behaves like an exponentially decaying function. Therefore, we consider the bell-shaped curve with shape parameter, $\kappa>1$. Without loss of generality, we focus on constant shape parameter, $\kappa=2$, 3 and 4 for simplicity. The simulation results in Section 5 demonstrate the concept that the increasing-then-decreasing scenario of delayed fault-detection process is more appropriate. That is, the bell-shaped factor can be well fit into the modeling the fault-correction processes. In the following, some classical bell-shaped functions that can be used as the delay-effect factor in Eq. (2) are proposed and the mean value functions are also derived. Furthermore, we observe that there are four types of correction processes that are classified by the growing speed of delay-effect factors and the results are listed in Table 1.

- **Gamma-type delay-effect factor**

Take the delay-effect factor, $\varphi(t)=ct^{\kappa-1}e^{-\frac{t}{\theta}}$, then we have

$$m(t)=a(1-e^{-bt}e^{b\varphi(t)})=a(1-e^{-bt}\exp(bct^{\kappa-1}e^{-t/\theta})) \quad (6)$$

- **Rayleigh-type delay-effect factor**

Take the delay-effect factor, $\varphi(t)=cte^{-(t/\theta)^2}$, then we have

$$m(t)=a(1-e^{-bt}e^{b\varphi(t)})=a(1-e^{-bt}\exp(bcte^{-(t/\theta)^2})) \quad (7)$$

- **Weibull-type delay-effect factor**

Take the delay-effect factor, $\varphi(t) = ct^{\kappa-1}e^{-(t/\theta)^{\kappa}}$, then we have

$$m(t) = a(1-e^{-bt}e^{b\varphi(t)}) = a(1-e^{-bt}\exp(bct^{\kappa-1}e^{-(t/\theta)^{\kappa}})). \quad (8)$$

Table 1: Classifications according to the speed of various delay-effect factors in all the testing phase.

Type	Model
Bell-shaped	Proposed model
Constant	Goel-Okumoto model
Unbounded increasing	Gompertz growth curve model
	Delayed S-shaped model
	Modified Duane model
	Weibull-type testing-effort function model
	Log-logistic growth curve model
Bounded increasing	Inflected S-shaped model
	Logistic growth curve model

IV. INTEGRATION OF DELAYED-TIME AND NON-DELAYED-TIME MODELS

Many existing software reliability models can be integrated under a general formulation. Model integration is helpful for the generalization of models without making many assumptions. From our studies, some model integration schemes are derived in the literature [21-23]. Recently, Hou *et. al.* [20] proposed a unified theory for several NHPP models, which applied the concepts of weighted arithmetic, geometric, harmonic, or quasi-arithmetic means to derive several existing and new NHPP models. Many existing software reliability models based on NHPP models can be derived as special cases of this general NHPP model. We quote the theorem in [20] as follows:

Theorem 2: Given a mean value function, $m(t)$, its corresponding fault-detection intensity, $\lambda(t)$ and let $\partial g(m(t))/\partial t = \lambda(t)\{g(a) - g(m(t))\}$, where g is a real-valued, strictly monotonic, and differentiable function. We have

$$m(t) = g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\}, \quad (9)$$

where a is the expected number of initial faults and $B(t) = \int_0^t \lambda(u)du$ [20]. □

In the following, we discuss a general continuous NHPP model from the viewpoint of delayed-time correction phenomenon and propose a general framework of the modeling of the fault-detection and fault-correction processes. First, according to the above assumption of delay-effect factor, the mean value function of delayed-time NHPP models can be described by $m(t) = a(1-e^{-bt}e^{b\varphi(t)})$, $a>0$, $b>0$. (i.e. Eq.(2)). Secondly, without loss of generality, we cite the general form of the mean value function of NHPP proposed by Hou *et. al.*, i.e. $m(t) = g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\}$, to be any desired type of the mean value function under discussion. The objective here is to give an appropriate delay-effect factor, $\varphi(t)$, so that Eq. (2) and Eq. (9) can be treated as identical. Therefore, we extend the unification theory of the general NHPP model to the delayed-time case by the following theorem.

Theorem 3: Given a general form of NHPP mean value function,

$$m(t) = g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\},$$

where g is a real-valued, strictly monotonic, and differentiable function. We have the general delayed-time form of the mean value function,

$$m(t) = a(1-e^{-bt}e^{b\varphi(t)}) \quad (10)$$

where

$$\varphi(t) = t + \frac{1}{b}\ln\{1 - \frac{1}{a}g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\}\}$$

Proof:

We know that g is a real-valued, strictly monotonic, and differentiable function. It implies that the inverse function g^{-1} exists. Let both right sides of Eq. (9) and Eq. (10) be identical, then

$$e^{-bt}e^{b\varphi(t)} = 1 - \frac{1}{a}g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\}, \text{ i.e.}$$

$$\varphi(t) = t + \frac{1}{b}\ln\{1 - \frac{1}{a}g^{-1}\{g(a) + [g(m(0)) - g(a)]\exp(-B(t))\}\}.$$

□

From the above theorem, we know that even other existing software reliability models based on NHPP can also be treated as special cases of this general delayed-time model. Namely, we may specify both functions $g(x)$ and $B(t)$ in Eq. (9), and then solve them for $m(t)$ to get a new model. On the other hand, we can also specify the delay-effect factor, $\varphi(t)$, in Eq. (10) and solve its corresponding delayed-time form of mean value function to get another new delayed-time correction process model. Furthermore, based on various functions for $g(x)$ in Eq. (10), we can derive its corresponding $\varphi(t)$. Specifically, we focus on identical function (i.e., $g(x) = x$), inverse function (i.e., $g(x) = \frac{1}{x}$), and logarithm function (i.e., $g(x) = \ln x$). Table 2 shows the relationship between various models that can be derived by choosing proper $\varphi(t)$, $g(x)$ and $B(t)$.

From the above derivation, we make the following observations:

1. The general form of the classical NHPP mean value function that is described in Eq. (9), i.e. assuming the correction process is perfect, is identical to the general delayed-time form of the mean value function described in Eq. (2).
2. By deriving the general NHPP model, many existing NHPP models for software reliability can be derived as special cases of this general framework.

Table 2: Classifications according to the speed of various delay-effect factors in all the testing phase

$m(t)$	$g(x)$	$\varphi(t)$	$B(t)$
$a(1-ke^{-B(t)})$	x	$t-\frac{B(t)}{b}+\ln k$	$B(t)$
G-O model	x	0	bt
Delayed S-shaped model	x	$\frac{1}{b}\ln(1+bt)$	$bt-\ln(1+bt)$
Inflected S-shaped model	x	$\frac{1}{b}\ln\frac{1+c}{1+ce^{-bt}}$	$bt+\ln\frac{1+ce^{-bt}}{1+c}$
Modified Duane model	x	$\frac{\gamma}{b}\ln(\frac{\beta}{\beta+t})+t$	$\gamma\ln\frac{\beta+t}{\beta}$
$a\frac{1}{1+ke^{-B(t)}}$	$\frac{1}{x}$	$\frac{1}{b}\ln\frac{k}{1+ke^{-B(t)}}$	$B(t)$
Logistic growth curve model	$\frac{1}{x}$	$\frac{1}{b}\ln\frac{k}{1+ke^{-bt}}$	bt
Log-logistic growth curve model	$\frac{1}{x}$	$\frac{1}{b}\ln(\frac{e^{bt}}{1+\lambda^k e^{k\ln t}})$	$\ln\frac{ce^{bt}}{c+c(\lambda t)^k-bt}$
$ak^{e^{B(t)}}$	$\ln x$	$t+\frac{1}{b}\ln(1-ke^{-B(t)}$	$B(t)$
Gompertz growth curve model	$\ln x$	$t+\frac{1}{b}\ln(1-ke^{-bt})$	bt
Eq.(6)	x	$ct^{\kappa-1}e^{\frac{-t}{\theta}}$	$bt-bct^{\kappa-1}e^{\frac{-t}{\theta}}$
Eq.(7)	x	$cte^{-(\frac{t}{\theta})^2}$	$bt-cte^{-(\frac{t}{\theta})^2}$
Eq.(8)	x	$ct^{\kappa-1}e^{-(\frac{t}{\theta})^\kappa}$	$bt-ct^{\kappa-1}e^{-(\frac{t}{\theta})^\kappa}$

$m(t)$: The mean value function.
$\varphi(t)$: The delay-effect factor.
$g(x)$, $B(t)$: The notation in Eq. (9).

V. EXPERIMENTAL RESULTS

By using the Maximum Likelihood Estimates, the parameters of the proposed NHPP models can be determined. Furthermore, we adopt the evaluation criteria, the mean of squares of errors (MSE) [8], in the comparison of goodness-of-fit of the models. A lower MSE indicates better performance of fitting the real software data. The example real internet-distributed software is the XbaeMatrix widget, which is a reusable Motif widget that provides a spreadsheet display capability for X window programs [2]. During the testing period, about 89 software faults were discovered. Table 3 shows that the estimated parameters and MSE of the proposed models described in Eq. (6)-Eq. (8). Figure 1 depicts the observed curve and the fitted curves of the cumulative numbers of failures and Figure 2 shows the estimated time lag using the proposed models. Especially, Eq. (8) with $\kappa = 3$ has the smallest value of MSE (117.93). Besides, the MSE value in Eq. (6) with $\kappa = 2, 3$ or 4 or Eq. (7) is less than that in other models; therefore, we can conclude that the bell-shaped delay-effect factor model gives a better fit in this experiment

VI. CONLUSIONS

In this paper, we have proposed new delayed-time NHPP models by incorporating the bell-shaped delay-effect factor to model the fault detection and correction processes. Specifically, the fault-correction process is modeled as a delayed fault-detection process and it lags the detection process by a time-dependent delay of a bell-shaped curve. Furthermore, we have proposed a general delayed-time NHPP model based on the delay-effect factor, which enables us to derive existing and new delayed-time NHPP models. Under this framework, an existing NHPP model can be reinterpreted as a new delayed-time NHPP model. The new delayed-time NHPP models can capture constant and non-decreasing time lag during software testing process, whereas the proposed general model can capture the increasing-then-decreasing scenario to reflect the human learning ability. A key factor of the proposed models is the "delay-effect factor", which measures the expected time-lag in correcting the detected faults during software development. In particular, we adopt the bell-shaped curves, such as the Gamma curve, the Rayleigh curve, or the Weibull curve, to fit the various types of fault removal processes. Furthermore, the proposed models can be extended to model other fault removal scenarios by introducing different curves. Experiments on internet-distributed data set have been performed. The results show that the proposed new models give a better performance in estimating the number of initial faults, and they also indicate a better goodness-of-fit in terms of the mean of squares criterion. On the other hand, based on the delay-effect factor view of the NHPP, we can eliminate the unrealistic assumption that detected faults are corrected immediately. It is noted that more resources are required for the determination of extra parameters in these new models. However, this overhead is minimal as the parameter estimation process is fully automated

Table 3: Comparison results and estimated parameters of the proposed models.

Model	a (1)	b	c	θ	*MSE*(2)
Eq. (7)	96.45	0.117	0.028	7	143.75
Eq. (6) with $\kappa=2$	94.5	0.155	0.066	19	154.34
Eq. (6) with $\kappa=3$	94.24	0.13	0.031	3	134.08
Eq. (6) with $\kappa=4$	95.17	0.125	0.013	2	135.97
Eq. (6) with $\kappa=3$	95.64	0.12	0.0091	6	117.93
G-O Model [2]	105.06	0.0939	NA	NA	172.03

(1) The expected number of initial faults.
(2) The mean of squares of errors.

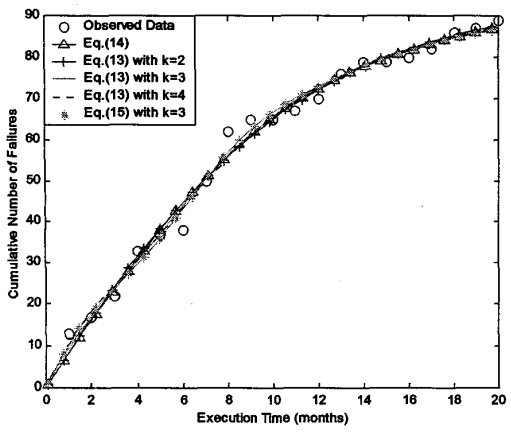

Figure 1: Observed/estimated curves of MVF.

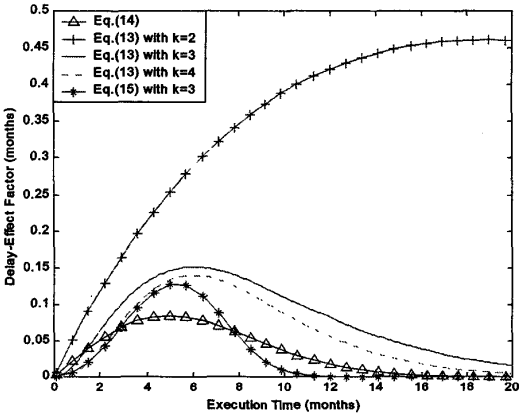

Figure 2: Delay-effect factor vs. execution time.

REFERENCES

[1] John D. Musa, "Software Reliability Engineering for Mobile Code," *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 182, 1998.

[2] Russell A. Fink, "Reliability Modeling of Freely-Available Internet-Distributed Software," *Proceedings of the 5th International Symposium on Metrics*, pp. 101-104, 1998.

[3] Jeffrey Voas, "Does Software Reliability Engineering (SRE) Offer Any Benefits to Mobile Code's Security Woes?" *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 180, 1998.

[4] Anup K. Ghosh, "On Certifying Mobile Code for Secure Applications," *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 381, 1998.

[5] M. R. Lyu (1996). *Handbook of Software Reliability Engineering.* McGraw Hill.

[6] M. Xie and M. Zhao, "The Schneidewind Software Reliability Model Revisited, " *Proceedings of the 3th International Symposium on Software Reliability Engineering* (ISSRE'92), pp. 184-192, 1992.

[7] S. S. Gokhale, "Analysis of Software Reliability and Performance," *Ph.D. Dissertation*, Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, 1998.

[8] M. Ohba, " Software Reliability Analysis Models, " *IBM J. Res. Develop.*, Vol. 28, No. 4, pp. 428-443, July 1984.

[9] A. L. Goel and K. Okumoto, "Time-dependent Error-detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Trans. on Reliability*, Vol. R-28, pp. 206-211, 1979.

[10] S. Yamada, M. Ohba, and S. Osaki, "S-shaped Reliability Growth Modeling for Software error detection", *IEEE Trans. on Reliability*, vol. R-32, No. 5, pp. 475-478, 1983.

[11] C. Y. Huang, S. Y. Kuo and I. Y. Chen, "Analysis of a Software Reliability Growth Model with Logistic Testing-Effort Function," *Proceedings of the 8th International Symposium on Software Reliability Engineering* (ISSRE'97), pp. 378-388, Nov. 1997, Albuquerque, New Mexico. U.S.A.

[12] C. Y. Huang, J. H. Lo and S. Y. Kuo, "A Pragmatic Study of Parametric Decomposition Models for Estimating Software Reliability Growth," *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 111-123, Nov. 4-7. 1998, Paderborn, Germany.

[13] C. Y. Huang, J. H. Lo , S. Y. Kuo and Michael R. Lyu, "Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency," *Proceedings of the 10th International Symposium on Software Reliability Engineering* (ISSRE'99), pp. 62-72, 1999.

[14] B. Littlewood, "Rationale for a Modified Duane Model", *IEEE Trans. on Reliability*, vol. 33, pp. 428-443, 1984.

[15] S. Yamada, J. Hishitani, and S. Osaki, "Software-Reliability Growth with a Weibull test-effort : a Model & Application", *IEEE Trans. on Reliability*, Vol. 42, No. 1, pp. 100-106, 1993.

[16] N. F. Schneidewind, "Analysis of Error Processes in Computer Software," Sigplan Notices, Vol. 10, pp. 337-346, 1975.

[17] J. D. Musa, A. Iannino, and K. Okumoto (1987). *Software Reliability, Measurement, Prediction and Application.* McGraw Hill.

[18] J. D. Musa (1998). *Software Reliability Engineering: More Reliable Software, Faster Development and Testing.* McGraw-Hill.

[19] A. L. Goel, "Software Reliability Models : Assumptions, Limitations, and Applicability," *IEEE Trans. on Software Engineering*, Vol. 11, No. 12, pp. 1411-1423,1985.

[20] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "On a Unified Theory of Some Nonhomogeneous Poisson Process Models for Software Reliability," *Proceedings of International Conference on Software Engineering : Education & Practice*, pp. 60-67, 1998.

[21] N. Langberg and N. D. Singpurwalla, "A Unification of Some Software Reliability Models," *SIAM J. Sci. Comput.*, Vol. 6, No. 3, pp. 781-790, 1985.

[22] D. R. Miller, "Exponential Order Statistic Models for Software Reliability Growth ," *IEEE Trans. on Software Engineering*, Vol. 12, No. 1, pp. 12-24, 1986.

[23] M. Trachtenberg, "A General Theory of Software-Reliability Modeling," *IEEE Trans. on Reliability*, Vol. 39, No. 1, pp. 92-96, 1990.

[24] R. H. Hou, S. Y. Kuo and Y. P. Chang, "Applying Various Learning Curves to Hyper-Geometric distribution Software Reliability Growth Model," *Proceedings of the 5th International Symposium on Software Reliability Engineering* , pp. 7-16, Nov. 1994, Monterey, California.

[25] A. A. Abdel-Ghally, P.Y. Chan , and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. on Software Engineering*, SE-12(9), pp. 538-546, Sept.1989.

[26] J. D. Musa, Software Reliability Data', Report and Data Base Available from Data and Analysis Center for Software, Rome Air Development Center (RADC). Rome, NY.

[27] M. Zhao and M. Xie, "On the Log-Power NHPP Software Reliability Model , " *Proceedings of the 3th International Symposium on Software Reliability Engineering* , pp. 14-22, 1992.