Annals of Operations Research
https://doi.org/10.1007/s10479-018-2951-7

S.I.: STOCHASTIC MODELING AND OPTIMIZATION, IN MEMORY OF ANDRÁS PRÉKOPA

A software reliability model incorporating martingale process with gamma-distributed environmental factors

Mengmeng Zhu[1] · **Hoang Pham**[1]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
As the increasing application of software system in various industry, software reliability gains more attention from the researchers and practitioners in the past few decades. The goal of such an expanding application of software system is to continuously bring convenience and functionality in everyday life. Lots of environmental factors defined by many studies may have positive/negative impact on software reliability during the development process (Zhu et al. in J Syst Softw 109:150–160, 2015; Clarke and O'Connor in Inf Softw Technol 54(5):433–447, 2012; Zhu and Pham in J Syst Softw 1–18, 2017b). However, most existing software reliability models have not incorporated these environmental factors in the model consideration. In this paper, we propose a theoretic software reliability model incorporating the fault detection process is a stochastic process due to the randomness caused by the environmental factors. The environmental factor, *Percentage of Reused Modules*, is described as a gamma distribution in this study based on the collected data from industry. Open Source Software project data are included to demonstrate the effectiveness and predictive power of the proposed model.

Keywords Environmental factors · Percentage of Reused Modules · Gamma distribution · Martingale process · Software reliability growth model · Non-homogeneous Poisson process · Brownian motion · Standard Gaussian white noise

1 Introduction

Software system is embedded in most products applied in our daily life, such as consumer electronics, home appliances, network/grid systems, and computer systems (Rana et al. 2014). The smooth and friendly user-interface and increasing innovation of the embedded software system bring in more convenience and functionality in everyday life. At the same time,

✉ Mengmeng Zhu
qimengzhu@gmail.com

[1] Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ 08854, USA

Published online: 27 June 2018

software system is becoming more sophisticated to meet various performance from different perspectives.

As discussed in the recent publication (Bosch 2010), both the size of software system built and the deployment speed of the new-built software system continue to evolve at a very enormous speed. Such evolvement requires software practitioner to continuously develop new approaches to manage the consequent implication of the environments. One of the major transitions is most software developments have shifted their attention from building the system toward composing system from the existing open source, commercial and internal developed components. Under such transition, the goal will be concentrated on the creation and functionality. Secondly, the lean and agile development approaches are commonly applied in many software development processes nowadays. Small team, short development iteration, and frequent deployment are the main elements of this approach. The third trend is the increasing adoption of software ecosystem. The development of new functionality can be occurred outside of the platform, not limited in the organization. App-store styled approaches are introduced by many companies to provide this feature to the market (Harman et al. 2012; Ghazawneh and Henfridsson 2013; Basole and Karla 2011).

Two issues are brought to our attention associated with these transitions in software industry: *Software Reliability* and *Environmental Factors* in development process. High software reliability must be demonstrated for each software release to assure it functions as designed in the operation environment (Palviainen et al. 2011). How to control and measure software quality is the most challenging issue faced by software industry (Hsu et al. 2011). Developing software and protecting against software failure contribute to the largest expenses of software industry (Chang and Liu 2009). Many software measurement metrics are developed to properly address software quality before its release. One of the most applied software measurement metrics is software reliability. Software reliability, not only helps manager with quantifying system behaviors, but allocating testing resource and optimizing release time. The commonest method to evaluate software reliability is to apply Software Reliability Growth Model (SRGM).

Since 1970s, many non-homogeneous Poisson process (NHPP) software reliability growth models in consideration of different assumptions have been proposed for the estimation of the number of remaining faults, failure rate, and software reliability for software products given a specific time horizon (Hsu et al. 2011; Goel and Okumoto 1979; Musa 1975; Kapur et al. 2011; Yamada et al. 1986; Zhu and Pham 2016; Ahmadi et al. 2016; Pachauri et al. 2015; Sukhwani et al. 2016; Yamada 2014; Jin and Jin 2014; Inoue et al. 2016; Fiondella and Zeephongsekul 2016; Zhu and Pham 2017a; Yamada and Tamura 2016; Zachariah 2015b; Özdamar and Alanya 2001). For example, incorporating testing effort into software reliability model has received lots of attention these years. Huang et al. (2007) incorporated logistic testing effort function into both exponential-type and S-type NHPP models. Lin and Huang (2008) discussed the multiple change-points into the flexible Weibull-type time dependent testing effort function. Later, Inoue et al. (2016) developed a bivariate software reliability growth model considering the uncertainty of the change-point. Additionally, the optimal release time and resource allocation is studied by many researchers (Xie and Hong 1998; Huang and Lyu 2005a; Lo et al. 2005; Huang and Lo 2006; Yang and Xie 2000; Huang and Lyu 2005b; Zachariah 2015a). These theoretical software reliability models have been effectively proven in a wide-range real application in industry and research institution (Almering et al. 2007; Jeske and Zhang 2005; Sukhwani et al. 2016). However, none of them has discussed the impact of environmental factor on software reliability during software development/operation and how to incorporate environmental factor into the modeling.

Zhang and Pham (2000) firstly defined 32 environmental factors during the software development process. These environmental factors not only included each phase of software development practice, but considered human nature, teamwork and the interactions amongst human, software system and hardware system. Later, Zhu et al. (2015) conducted another comparison study to address the impact level of each environmental factor on software reliability assessment and provided up-to-date critical environmental factors to software practitioners.

Only a few literatures incorporated the randomness of the environmental factors in software reliability model. Teng and Pham (2006) proposed a generalized random field environment software reliability model that linked both the testing phase and operation phase. The fault detection rate for testing and operation phase is not constant. Hsu et al. (2011) integrated fault reduction factor into software reliability modeling, where fault reduction factor is defined as the ratio of net fault reduction to the failure experienced. Recently, Pham (2014) considered the uncertainty of the operation environment into software reliability model.

In this paper, we consider one of the top 10 critical environmental factors (Zhu et al. 2015), *Percentage of Reused Modules* (PoRM), to be a random variable which has random effect on software fault detection rate. The real data collected from several industries is applied to obtain the distribution of PoRM in Sect. 2. We then introduce the martingale framework, specifically, Brownian motion and white noise process in the stochastic fault detection process to reflect the impact resulting from the randomness of environmental factor. The proposed NHPP software reliability model incorporating these considerations is presented in Sect. 3. Parameter estimation and model comparison criteria are discussed in Sect. 4. Two Open Source Software project data are analyzed in Sect. 5 to illustrate the effectiveness of the propose model. The reliability prediction and the significance of PoRM are also discussed in Sect. 5. Specifically, we are interested in the predictive accuracy comparison of the proposed model with and without considering PoRM. In Sect. 6, we conclude this study, discuss the directions for the future research and the limitations of the current study.

Notation

$m(t, \eta)$: expected number of software failures by time t, also mean value function
$N(t)$: total fault content function in the software by time t
$h(t, \eta)$: time-dependent fault detection rate per unit of time
$G(t, \eta)$: impact of *Percentage of Reused Modules* (PoRM) on failure detection rate per unit of time
$f(\eta)$: probability density function of PoRM
$M(t, w)$: martingale with respect to the filtration $(\mathbb{F}_t, t \geq 0)$
$\dot{M}(t, w)$: derivative of $M(t, w)$ with respect to time t
$v(t)$: measures the impact of time on PoRM
$F^*(s)$: Laplace transform of $f(\eta)$
θ, γ: parameters of gamma distribution
η: random variable, which represents PoRM
λ: coefficient along with $G(t, \eta)$
b, c, a, k: coefficient in function $h(t)$, $v(t)$, and $N(t)$
y_i: observed number of software failures at time t_i

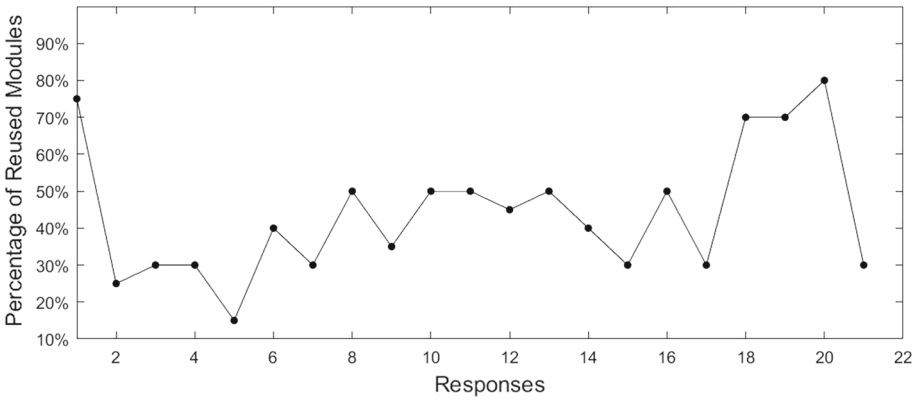

Fig. 1 Data collection of *Percentage of Reused Modules* (PoRM)

2 Environmental factor (PoRM) distribution

Although some existing software reliability models have considered the environmental factors in the modeling, most of them assume the distribution of environmental factor based on their knowledge and model consideration. For example, Teng and Pham (2006) described the random environmental factor is 1 in in-house testing phase and a random variable in operation phase. They assumed that this random variable follows gamma distribution or beta distribution in reliability models. But, there is no real data to support this assumption.

In this study, the definition of *Percentage of Reused Modules* (PoRM), adopted from reference (Zhang and Pham 2000; Zhang et al. 2001; Zhu and Pham 2017b), is presented as follows

$$PoRM = \frac{S_0}{S_N + S_0} \tag{1}$$

where S_0 represents kiloline of code for the existing modules and S_N represents kiloline of code for new modules.

In this section, we employ the real data collected from several industries to illustrate the distribution of PoRM. Participants were asked to provide *Percentage of Reused Modules* in their industry based on the relevant working experience. All the participants are currently working in IT Department in different industry including computer software, banking, semiconductor, online retailing, IT service and research institution or working on software development in high-tech company in favor of the validity and reliability of the survey response. The collected *Percentage of Reused Modules* (PoRM) are shown in Fig. 1.

Eight different distributions commonly applied to model environmental factors are adopted in the model comparison to estimate the distribution of PoRM. Maximum likelihood estimation (MLE) is applied to compare the effectiveness of each model. The log-likelihood values for all models are shown in Table 1.

Gamma distribution will be employed to model the distribution of PoRM. The parameters of gamma distribution are also obtained from model comparison: $\theta = 14.726$, $\gamma = 6.487$.

$$f(\eta) = \frac{\theta^{\gamma}\eta^{\gamma-1}e^{-\theta\eta}}{\Gamma(\gamma)} \tag{2}$$

Table 1 Log-likelihood value comparison

Normal dist.	Weibull dist.	Beta dist.	Exponential dist.	Gamma dist.	Inverse Gaussian dist.	Log-logistic dist.	Lognormal dist.
7.041	7.639	7.830	−3.782	8.174	8.002	7.739	8.015

3 Software reliability modeling

3.1 Related works

We first provide the definition of software fault and software failure in order to clearly address the software reliability models discussed below. Software fault is caused by an incorrect step, process, or data definition in a computer program (Radatz et al. 1990). Failure is defined as a system or component is not able to perform its required functions within specified performance requirements. Software failure is the manifestation of software fault (Radatz et al. 1990).

Many NHPP software reliability models have been proposed for the past four decades to address different assumptions. The failure processes are described by NHPP property with the mean value function (MVF) at time t, $m(t)$, and the failure intensity of the software, $\lambda(t)$, which is also the derivative of the mean value function. Most existing NHPP software reliability models are developed based on the model given as follows

$$\frac{d}{dt}m(t) = h(t)(N(t) - m(t)) \tag{3}$$

where $m(t)$ is the expected number of software failures by time t, $N(t)$ is the total number of fault content by time t, and $h(t)$ is the time-dependent fault detection rate per unit of time. The underlying assumption of Eq. (3) is the failure intensity is proportional to the residual fault content in the software. Lots of NHPP software reliability models are developed based on Eq. (3). Some examples are given in Table 2.

The software reliability models presented in Table 2, along with other deterministic models, have not taken into account the effect of randomness from the development/operation environment. To capture the effect of random environments, a stochastic process can be incorporated in the fault detection process by imposing $h(t)$ to be $h(t, \eta)$, where η is the random effect. The Eq. (3) will be described as

$$\frac{d}{dt}m(t, \eta) = h(t, \eta)(N(t) - m(t, \eta)) \tag{4}$$

For example, in Pham (2014), the author considered

$$h(t, \eta) = h(t)\eta, N(t) = N \tag{5}$$

where η is a random variable and follows gamma distribution with parameter α and β. By substituting Eq. (5) to Eq. (4), the explicit solution of the mean value function can be expressed as (Pham 2014)

$$m(t) = N\left(1 - \left(\frac{\beta}{\beta + \int_0^t h(s)ds}\right)^{\alpha}\right) \tag{6}$$

Table 2 Some existing NHPP software reliability models

NHPP model	Functions	Mean value function
Geol–Okumoto	$h(t) = b, N(t) = a$	$m(t) = a\left(1 - e^{-bt}\right)$
Delayed S-shaped	$h(t) = \frac{bt^2}{1+bt}, N(t) = a$	$m(t) = a\left(1 - (1+bt)e^{-bt}\right)$
Inflection S-shaped	$h(t) = \frac{b}{1+\beta e^{-bt}}, N(t) = a$	$m(t) = \frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$
Yamada imperfect debugging model	$h(t) = b, N(t) = a(1+\alpha t)$	$m(t) = a\left[1 - e^{-bt}\right]\left[1 - \frac{\alpha}{b}\right] + \alpha at$
PNZ model	$h(t) = \frac{b}{1+\beta e^{-bt}}, N(t) = a(1+\alpha t)$	$m(t) = \frac{a\left[(1-e^{-bt})\left(1-\frac{\alpha}{b}\right)+\alpha t\right]}{1+\beta e^{-bt}}$
Pham–Zhang IFD	$h(t) = \frac{b^2 t}{1+bt} + \frac{d}{1+dt}$ $N(t) = a$	$m(t) = a - ae^{-bt} \times \left(1 + (b+d)t + bdt^2\right)$

The above formulation, $h(t, \eta) = h(t)\eta$, proposed in Pham (2014) is referred as dynamic multiplicative noise model in recent publication from Pham and Pham (2017). They (Pham and Pham 2017) also proposed a dynamic additive noise model shown as follows

$$h(t, w) = h(t) + \dot{M}(t, w) \tag{7}$$

where $\dot{M}(t, w)$ denotes the derivative of M with respect to time t. $M(t)$ is defined as a martingale with respect to the filtration $(\mathbb{F}_t, t \geq 0)$in this function. One worth-noting martingale property (Mikosch 1998; Mörters and Peres 2010) applied in Eq. (7) is

$$E\left(\int_v^t h(s, w)ds\right) = \int_v^t h(s)ds \tag{8}$$

Brownian motion (Mikosch 1998) is also a martingale. Let $\{B(t) : t \geq 0\}$ be Brownian motion, Mikosch (1998) mentioned that $\{B(t) : t \geq 0\}$ and $\left\{B^2(t) - t : t \geq 0\right\}$ are martingale with respect to the nature filtration $(\mathbb{F}_t, t \geq 0)$. For the model consideration in this paper, we also choose $M(t)$ to be Brownian motion since Brownian motion is deeply researched by many literatures.

A stochastic process $\{B(t) : t \geq 0\}$ is called Brownian motion (Mörters and Peres 2010) with start in $x \in \mathbb{R}$ if

1. $B(0) = x$.
2. The process has independent increments. For example, the increments $B(t_n) - B(t_{n-1}), B(t_{n-1}) - B(t_{n-2}), \ldots, B(t_2) - B(t_1)$ are independent for all time $0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$.
3. The increments $B(t+s) - B(t)$ are normally distributed, for all $t \geq 0, s \geq 0$.

If $B(0) = 0$, i.e. the motion is started from the origins, this is called standard Brownian motion.

3.2 Proposed NHPP software reliability model

The proposed NHPP software reliability model in this study incorporates both the dynamic multiplicative and additive noise characteristics discussed in Sect. 3.1. Meanwhile, one of the top 10 environmental factors from the recent comparison survey study (Zhu et al. 2015), modeled as gamma distribution with parameter θ and r estimated from real-collected data from various industries, is incorporated in the model development. In sum, environmental factor (PoRM), described as gamma distribution from real data, and the randomness caused by PoRM, illustrated by the martingale, specifically, Brownian motion, are taken into account for the proposed software reliability model.

The assumptions for the proposed model are given as follows

1. The fault removal process follows the non-homogeneous Poisson distribution.
2. The failure of software system is subject to the manifestation of the remaining faults in the software program.
3. All faults in the software are independent.
4. The failure intensity is proportional to the remaining faults in the software.
5. The environmental factor, *Percentage of Reused Modules* (PoRM), is described as gamma distribution based on the real data collected from various industry, as discussed in the Sect. 2. The impact of PoRM is explained by an additive portion along with the traditional fault detection process.
6. The randomness caused by the environmental factor is explained as a martingale, particularly, Brownian motion in this study.

The proposed NHPP software reliability model is given as

$$\frac{d}{dt}m(t,\eta) = \left[h(t) + \lambda G(t,\eta) + \dot{B}(t)\right][N(t) - m(t,\eta)] \tag{9}$$

$$m(0) = 0 \tag{10}$$

where $m(t,\eta)$ is the expected number of failures by time t, $h(t)$ is the fault detection rate per unit of time without considering the effect of environmental factor, η is a random variable and represents the environmental factor, PoRM, $G(t,\eta)$ is a time-dependent function and represents the impact of PoRM on failure detection rate per unit of time, λ is the coefficient along with $G(t,\eta)$, and $\dot{B}(t)$ denotes a standard Gaussian white noise, specifically

$$\frac{d}{dt}B(t) = \dot{B}(t) \tag{11}$$

where $B(t)$ is a Brownian motion.

The covariance of $\dot{B}(t)$ is

$$E\left(\dot{B}(t)\dot{B}(u)\right) = \delta(u-t), 0 < t < u \tag{12}$$

From (Pham and Pham 2017), similarly, we can obtain the general solution for Eq. (9) as follows

$$m(t,\eta) = N(t) - N(0)e^{-\int_0^t (h(s)+\lambda G(s,\eta)+\dot{B}(s))ds} - \int_0^t e^{-\int_u^t (h(s)+\lambda G(s,\eta)+\dot{B}(s))ds} N'(u)du \tag{13}$$

The detailed derivation can be found in "Appendix I".

Since $\dot{B}(t)$ denotes a standard Gaussian white noise, we can also have

$$\int_0^t \big(h(s)+\lambda G(s,\eta)+\dot{B}(s)\big)ds = \int_0^t (h(s)+\lambda G(s,\eta))ds + B(t) - B(0)$$
$$= \int_0^t (h(s)+\lambda G(s,\eta))ds + B(t) \tag{14}$$

The mean value function can be expressed as

$$\bar{m}(t) = N(t) - N(0)e^{-\int_0^t h(s)ds} e^{\frac{t}{2}} e^{-\int_0^t \lambda G(s,\eta)ds} - \int_0^t e^{-\int_u^t h(s)ds} e^{\frac{t-u}{2}} e^{-\int_u^t \lambda G(s,\eta)ds} N'(u)du \tag{15}$$

Let

$$G(t,\eta) = \eta v(t) \tag{16}$$

where $v(t)$ is a time-dependent function, measures the impact of time on PoRM.

The environmental factor, η, has a gamma distribution with the parameter θ and r.

$$f(\eta) = \frac{\theta^\gamma \eta^{\gamma-1} e^{-\theta\eta}}{\Gamma(\gamma)} \tag{17}$$

We apply the expectation on Eq. (15) with respect to η. The mean value function can be obtained

$$\bar{m}_\eta(t) = N(t) - N(0)e^{-\int_0^t h(s)ds} e^{\frac{t}{2}} \int_0^\infty e^{-\int_0^t \lambda\eta v(s)ds} f(\eta)d\eta$$
$$- \int_0^\infty \int_0^t e^{-\int_u^t h(s)ds} e^{\frac{t-u}{2}} e^{-\int_u^t \lambda\eta v(s)ds} N'(u) f(\eta)du d\eta$$
$$= N(t) - N(0)e^{-\int_0^t h(s)ds} e^{\frac{t}{2}} \int_0^\infty e^{-\eta\int_0^t \lambda v(s)ds} f(\eta)d\eta$$
$$- \int_0^t N'(u) e^{-\int_u^t \left(h(s)-\frac{1}{2}\right)ds} \left[\int_0^\infty e^{-\eta\int_u^t \lambda v(s)ds} f(\eta)d\eta\right] du \tag{18}$$

Compute the equations with Laplace transform

$$\int_0^\infty x e^{-sx} f(x)dx = -\frac{dF^*(s)}{ds} \tag{19}$$

The Laplace transform of gamma probability density function is

$$F^*(s) = \left[\frac{\theta}{\theta+s}\right]^\gamma \tag{20}$$

Therefore

$$\begin{aligned}\bar{m}_\eta(t) &= N(t) - N(0)e^{-\int_0^t h(s)ds}e^{\frac{t}{2}}F^*\left(\int_0^t \lambda v(s)ds\right) - \int_0^t N'(u)e^{-\int_u^t \left(h(s)-\frac{1}{2}\right)ds}F^*\left(\int_0^t \lambda v(s)ds\right)du \\ &= N(t) - N(0)e^{-\int_0^t h(s)ds}e^{\frac{t}{2}}\left[\frac{\theta}{\theta+\int_0^t \lambda v(s)ds}\right]^\gamma - \int_0^t N'(u)e^{-\int_u^t \left(h(s)-\frac{1}{2}\right)ds}\left[\frac{\theta}{\theta+\int_0^t \lambda v(s)ds}\right]^\gamma du \\ &= N(t) - N(0)e^{-\int_0^t h(s)ds}e^{\frac{t}{2}}\left[\frac{\theta}{\theta+\int_0^t \lambda v(s)ds}\right]^\gamma - \left[\frac{\theta}{\theta+\int_0^t \lambda v(s)ds}\right]^\gamma \int_0^t N'(u)e^{-\int_u^t \left(h(s)-\frac{1}{2}\right)ds}du\end{aligned} \tag{21}$$

The Eq. (21) provides a general solution for the proposed NHPP software reliability model in consideration of PoRM and the randomness caused by this environmental factor. Different formulation for $h(t)$, $v(t)$ and $N(t)$ with respect to different assumptions can be plugged in Eq. (21) to obtain the final solution. In this paper, we apply the formulation as follows.

Let

$$h(t) = \frac{b}{1+ce^{-bt}} \tag{22}$$

$$v(t) = e^{-at} \tag{23}$$

$$N(t) = \frac{1}{k}e^{kt} \tag{24}$$

As described in Melo et al. (1998), the application process of the reused module can be described as follows: (1) Reused module selection. (2) The adaption of the reused module to the new objective. (3) The integration to the new-developed software product. The impact of PoRM on the software fault detection rate will decrease as software development moves to the later phase. Thus, we use an exponentially decreasing function e^{-at} to represent $v(t)$ in this study. Meanwhile, $\frac{1}{k}e^{kt}$ is employed to represent $N(t)$ due to $\frac{1}{k}e^{kt}$ is a monotonically increasing function, which can represent the nature growth of the software failures, and the inspiration from reference (Pham and Pham 2017).

Substitute Eqs. (22–24) to Eq. (21), the mean value function can be expressed as

$$\begin{aligned}m(t,\eta) &= \frac{1}{k}e^{kt} - \frac{1}{k}\frac{c+1}{c+e^{bt}}e^{\frac{t}{2}}\left[\frac{\theta}{\theta+\frac{\lambda}{a}\left(1-e^{-at}\right)}\right]^\gamma \\ &\quad - \left[\frac{\theta}{\theta+\frac{\lambda}{a}\left(1-e^{-at}\right)}\right]^\gamma \frac{e^{\frac{t}{2}}}{c+e^{bt}}\int_0^t N'(u)\left[ce^{-\frac{u}{2}}+e^{\left(b-\frac{1}{2}\right)u}\right]du \\ &= \frac{1}{k}e^{kt} - \frac{e^{\frac{t}{2}}}{c+e^{bt}}\left[\frac{\theta}{\theta+\frac{\lambda}{a}\left(1-e^{-at}\right)}\right]^\gamma \\ &\quad \left(\frac{c+1}{k} - \frac{c}{k-\frac{1}{2}}e^{\left(k-\frac{1}{2}\right)t} - \frac{1}{b+k-\frac{1}{2}}e^{\left(b+k-\frac{1}{2}\right)t} + \frac{c}{k-\frac{1}{2}} + \frac{1}{b+k-\frac{1}{2}}\right)\end{aligned} \tag{25}$$

Given the definition of $m(t)$ is the expected number of software failures by time t, we do know $m(t)$ is an increasing function and we are also able to show $when\ t \to \infty, m(t) \to \infty$ for Eq. (25) based on the numerical example.

We believe the behavior of Eqs. (21) and (25) mainly depends on the formulation of $N(t)$. For example,

1. If $N(t)$ is a constant, then $m(t)$ converges to this constant.
2. If $N(t)$ is a time-dependent function and can converge to a finite number, then $m(t)$ converges to this finite number.
3. If $N(t)$ is a time-dependent function and cannot converge, then $m(t)$ cannot converge.

But in this paper, we are not going to cover the mathematical proof for the above description. The investigation of the function behavior will be further studied in the future research.

4 Parameter estimation and criteria

4.1 Parameter estimation

In practice, parameter estimation can be achieved by applying least square estimate (LSE) and maximum likelihood estimate (MLE). For example, minimizing Eq. (26) or maximizing Eq. (27).

$$S = \sum_{i=1}^{n} (m(t_i) - y_i)^2 \tag{26}$$

$$LLF = \sum_{i=1}^{n} (y_i - y_{i-1}) \log\left[m(t_i) - m(t_{i-1})\right] - m(t_n) - \sum_{i=1}^{n} \log(y_i - y_{i-1})! \tag{27}$$

where y_i is the observed number of failures at time t_i. We apply LSE to minimize Eq. (26) to estimate the parameters. Since the parameters of gamma distribution have already been estimated from the real data, thus, we need to estimate other five unknown parameters for Eq. (25), which are k, b, c, λ, and a. The genetic algorithm (GA) is employed in order to solve the optimization function, as given in Eq. (26). Matlab and R software are used to solve the optimization function and estimate parameters.

4.2 Comparison criteria

1. Mean squared error (MSE)

$$MSE = \frac{\sum_{i=1}^{n} (\widehat{m}(t_i) - y_i)^2}{n - N} \tag{28}$$

where n is the total number of observations, N represents the number of unknown parameters in each model, and y_i is the observed number of failures at time t_i. The mean squared error measures the distance of model estimate from the observed data.

2. Predictive-ratio risk (PRR) and predictive power (PP) (Pham 2007)

$$PRR = \sum_{i=1}^{n} \left(\frac{\widehat{m}(t_i) - y_i}{\widehat{m}(t_i)} \right)^2 \tag{29}$$

$$PP = \sum_{i=1}^{n} \left(\frac{\widehat{m}(t_i) - y_i}{y_i} \right)^2 \tag{30}$$

The predictive-ratio risk and predictive power are calculated to compare the predictive performance of different software reliability models. The predictive-ratio risk measures the distance of the model estimate from the actual data against the model estimate by assigning a larger penalty to the model that has underestimated the cumulative number of failures. The predictive power measures the distance of the model estimate from the actual data against the actual data.

3. Variation

The variation is defined as (Huang and Kuo 2002)

$$Variation = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \widehat{m}(t_i) - Bias)^2} \tag{31}$$

where

$$Bias = \frac{1}{n}\sum_{i=1}^{n}(\widehat{m}(t_i) - y_i) \tag{32}$$

For all the criteria discussed above, the smaller of the criteria, the better fit of the model.

As the research problems become more complex, it is very nature to contain lots of parameters. The proposed software reliability model has five parameters, which is still reasonable to be applied in industrial application. We have also chosen some model evaluation criteria, for example, variation, to assign the penalty for the model contains more parameters and address possible overfitting issue. Additionally, we have applied two datasets collected from real world to compare the predictability as well as the practical applicability of the proposed model with other software reliability models in Sect. 5.

5 Applications

In the following experiments, we choose two applications, DS1 and DS2, to validate the proposed NHPP software reliability model and compare the performance with other existing software reliability models. The comparison criteria are discussed in the above section. DS1 and DS2 are both collected from Open Source Software (OSS) project. Open Source Software has attracted significant attention in the past decade. Some report shows that a few major Open Source Software products have surpassed their commercial counterparts in terms of the market share and quality evaluation (Wheeler 2007). Not only individuals are attracted by the features of Open Source Software, but many software companies and government-supported organizations (Zhou and Davis 2005). A research study conducted by *CIO Magazine* (Cosgrove 2003) found that IT community is growing better by using open source development model and open source software will dominate as the Web server application platform and server operating system. The majority of the companies are using open source today for web development. To align with the new transitions in software development, Open Source Software project data are employed to illustrate the performance of the proposed model.

5.1 Dataset 1 (DS1)

DS1 is extracted from Apache Open Source Software project. It was collected from Sep 2010 to April 2013. The collected data are described in Table 3. The column named *Failure* in

Table 3 DS1 failure data

Time unit	Failures	Cumulative failures	Time unit	Failures	Cumulative failures	Time unit	Failures	Cumulative failures
1	6	6	12	4	66	23	6	102
2	6	12	13	0	66	24	22	124
3	6	18	14	4	70	25	3	127
4	8	26	15	5	75	26	1	128
5	13	39	16	5	80	27	1	129
6	6	45	17	2	82	28	0	129
7	8	53	18	10	92	29	0	129
8	2	55	19	1	93	30	0	129
9	3	58	20	1	94	31	4	133
10	3	61	21	2	96	32	3	136
11	1	62	22	0	96			

Table 3 represents the number of software failures detected between time unit $t-1$ and t. The column named *Cumulative Failures* in Table 3 represents the cumulative software failure by time t. The parameter estimate and model comparison are presented in Table 4. In this dataset, we use the first 24 time units to estimate the parameters.

As presented in Table 4, we can see that the proposed model has the smallest MSE, PRR, and Variation. It is worth noting that MSE is the most critical criteria in terms of model selection. The PP value for the proposed model is 0.311, which is just slightly larger than the smallest PP value 0.228, however, all other three criteria for G-O model is significantly larger than the proposed model. Thus, we conclude that the proposed model has the best fit. Figure 2 also shows the comparison between the actual failure data and the predicted failure data from all the models discussed in Table 4. The proposed model yields very close fittings and predictions on software failure.

Other software reliability measures are also calculated to provide a comprehensive understanding of the prediction power for the proposed model. In this study, we are interested in the estimated software failures for each time unit, which can be obtained by $m(t)-m(t-1)$, and the estimated time to detect all the 136 actual software failures that have already appeared in the program, as seen in Table 3. Figure 3 shows the estimated software failures for 16 time units, ranging from time unit 25–40, which gives an estimate for software tester in terms of the number of software failures occurred in the operation field during each time unit and further helps software multiple release planning. Moreover, the estimated time unit to detect all the 136 actual software failures is 38.40 based on the proposed model. Software tester can use this information to estimate how much time will be spent on one project and assign the corresponding testing resource. In sum, these measures can help software testing team to optimally assign the available testing resource to each ongoing project, decide when to stop testing, and plan software multiple release.

5.2 Dataset 2 (DS2)

DS2 is also one of Apache Open Source Software project data. It was collected from Feb 2009 to Feb 2014. 33 sets of failure data are presented in Table 5. The column named *Failure*

Table 4 DS1 parameter estimate and model comparison

Model	MVF	MSE	PRR	PP	Variation	Parameter estimate
G-O model	$m(t) = a\left(1 - e^{-bt}\right)$	36.561	0.315	0.228	6.076	$\widehat{a} = 201.250$ $\widehat{b} = 0.033$
Inflection S-shaped model	$m(t) = \frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	68.326	0.789	0.483	7.992	$\widehat{a} = 150.030$ $\widehat{b} = 0.096$ $\widehat{\beta} = 1.830$
Delayed S-shaped model	$m(t) = a\left(1 - (1+bt)e^{-bt}\right)$	110.047	20.902	2.123	10.544	$\widehat{a} = 131.400$ $\widehat{b} = 0.144$
Yamada imperfect debugging model	$m(t) = a\left[1 - e^{-bt}\right]\left[1 - \frac{\alpha}{b}\right] + \alpha at$	60.193	0.401	0.300	74.941	$\widehat{a} = 185.180$ $\widehat{b} = 0.033$ $\widehat{\alpha} = 0.01$
PNZ model	$m(t) = \frac{a\left[\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha t\right]}{1+\beta e^{-bt}}$	54.515	0.523	0.333	6.795	$\widehat{a} = 161.010$ $\widehat{b} = 0.069$ $\widehat{\alpha} = 0.001$ $\widehat{\beta} = 0.93$
Pham–Zhang IFD	$m(t) = a - ae^{-bt} \times \left(1 + (b+d)t + bdt^2\right)$	143.253	40.461	2.665	11.076	$\widehat{a} = 143.045$ $\widehat{b} = 0.129$ $\widehat{d} = 0.001$
Proposed model	$m(t) = \frac{1}{k}e^{kt} - \frac{e^{\frac{t}{2}}}{c+e^{bt}}\left[\frac{\theta}{\theta+\frac{\lambda}{a}(1-e^{-at})}\right]^{\gamma}\left(\frac{c+1}{k} - \frac{c}{k-\frac{1}{2}}e^{\left(k-\frac{1}{2}\right)t} - \frac{1}{b+k-\frac{1}{2}}e^{\left(b+k-\frac{1}{2}\right)t} + \frac{c}{k-\frac{1}{2}} + \frac{1}{b+k-\frac{1}{2}}\right)$	35.173	0.254	0.311	5.390	$\widehat{k} = 0.014$ $\widehat{b} = 0.589$ $\widehat{c} = 0.039$ $\widehat{a} = 75.000$ $\widehat{\lambda} = 2.001$

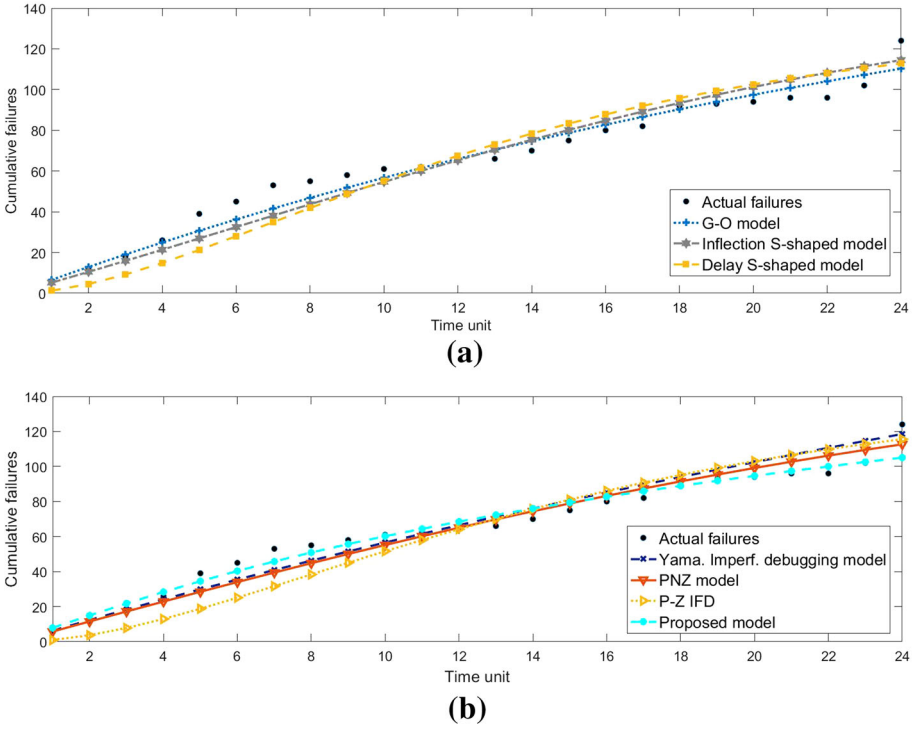

Fig. 2 Comparison of actual failure data and predicted failure data (DS1)

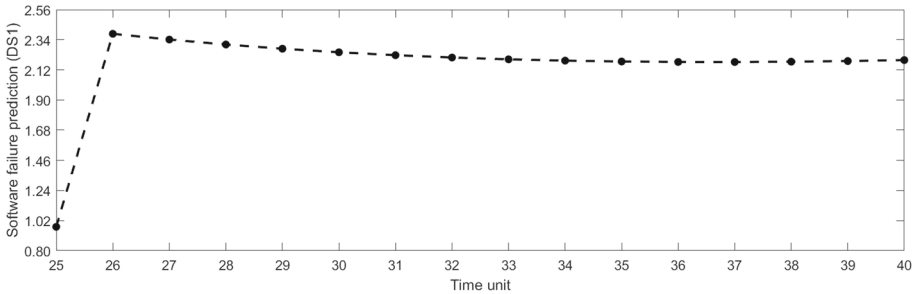

Fig. 3 Software failure prediction from time unit 25–40 (DS1)

in Table 5 represents the number of software failures detected between time unit $t-1$ and t. The column named *Cumulative Failures* in Table 5 represents the cumulative software failure by time t. The parameter estimate and model comparison are described in Table 6.

As seen from Table 6, the proposed model has the smallest MSE and Variation. Although PNZ model has the smallest PRR and PP, the MSE for PNZ model is much larger than the proposed model. Given MSE is the most critical for model selection, therefore, the best fitting model is still the proposed model. Figure 4 shows the comparison between the actual failure data and the predicted failure data from all the models discussed in Table 6. The proposed model subjects to a very close fittings and predictions on the cumulative software failures for DS2.

Table 5 DS2 failure data

Time unit	Failures	Cumulative failures	Time unit	Failures	Cumulative failures	Time unit	Failures	Cumulative failures
1	7	7	12	22	88	23	11	140
2	2	9	13	11	99	24	4	144
3	8	17	14	8	107	25	0	144
4	9	26	15	2	109	26	5	149
5	2	28	16	7	116	27	0	149
6	5	33	17	3	119	28	9	158
7	5	38	18	4	123	29	13	171
8	7	45	19	1	124	30	1	172
9	10	55	20	0	124	31	1	173
10	9	64	21	0	124	32	12	185
11	2	66	22	5	129	33	0	185

We also provide other software reliability measures for DS2. Figure 5 presents the estimated failures for each time unit, ranging from time unit 29 to 44. Moreover, the estimated time unit to detect all 185 software failures that have been already appeared in the program is 50.92.

5.3 Reliability prediction

Once the parameters are obtained, the software reliability within $(t, t + x)$ can be determined as

$$R(x|t) = e^{-[m(t+x)-m(t)]} \tag{33}$$

Figure 6 presents the reliability prediction for DS1 and DS2 by varying x from time unit 0 to 1.2. All other models do not take into consideration of environmental factor (PoRM) and a dynamic fault detection process. As a result, they cannot present reliability prediction well since OSS projects are influenced by the randomness caused by environmental factors significantly than traditional projects. Thus, we did not incorporate reliability prediction from other models.

During the testing phase, as the testing time increases, in general, software testing team can remove more software faults. In this case, software reliability will increase as the testing time increases. But after software released to operation phase (customers), the software reliability will decrease as the operation time increases if there is no updated version for the current release. First, it is impossible to remove all the software faults in the program due to the limitation of testing resource, programmer skill and constricted product development schedule (Zhu et al. 2015; Rana et al. 2014; Almering et al. 2007; Zhang et al. 2001; Zhu and Pham 2017a, b). Hence, even software product has already been released in the market, it still contains some software faults. Second, software users will also contribute incorrect/correct input during the operation phase, which can trigger the undetected software faults existed in the program. Thus, the total number of software failures will increase as the operation time increases if there is no updated version, then the software reliability tends to go zero in this case. The software reliability prediction shown in Fig. 6 also confirms this conclusion.

Table 6 DS2 Parameter estimate and model comparison

Model	MVF	MSE	PRR	PP	Variation	Parameter Estimate
G–O model	$m(t) = a\left(1 - e^{-bt}\right)$	136.477	1.285	3.462	11.892	$\widehat{a} = 181.250$ $\widehat{b} = 0.056$
Inflection S-shaped model	$m(t) = \frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	176.235	5.292	1.513	12.794	$\widehat{a} = 179.230$ $\widehat{b} = 0.193$ $\widehat{\beta} = 13.159$
Delayed S-shaped model	$m(t) = a\left(1 - (1+bt)e^{-bt}\right)$	67.922	27.778	1.536	8.221	$\widehat{a} = 200.090$ $\widehat{b} = 0.112$
Yamada imperfect debugging model	$m(t) = a\left[1 - e^{-bt}\right]\left[1 - \frac{\alpha}{b}\right] + \alpha a t$	69.510	0.625	1.180	103.480	$\widehat{a} = 230.250$ $\widehat{b} = 0.034$ $\widehat{\alpha} = 0.008$
PNZ model	$m(t) = \frac{a[\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha t]}{1+\beta e^{-bt}}$	90.902	0.372	0.418	8.809	$\widehat{a} = 300.130$ $\widehat{b} = 0.048$ $\widehat{\alpha} = 0.001$ $\widehat{\beta} = 1.321$
Pham–Zhang IFD	$m(t) = a - ae^{-bt} \times \left(1 + (b+d)t + bdt^2\right)$	74.124	528.248	2.576	8.240	$\widehat{a} = 189.960$ $\widehat{b} = 0.134$ $\widehat{d} = 0.010$
Proposed model	$m(t) = \frac{1}{k}e^{kt} - \frac{e^{\frac{t}{2}}}{c+e^{bt}}\left[\frac{\theta}{\theta+\frac{\lambda}{a}(1-e^{-at})}\right]^{\gamma}\left(\frac{c+1}{k} - \frac{c}{k-\frac{1}{2}}e^{\left(k-\frac{1}{2}\right)t} - \frac{1}{b+k-\frac{1}{2}}e^{\left(b+k-\frac{1}{2}\right)t} + \frac{c}{k-\frac{1}{2}} + \frac{1}{b+k-\frac{1}{2}}\right)$	63.989	4.895	1.224	7.576	$\widehat{k} = 0.009$ $\widehat{b} = 0.626$ $\widehat{c} = 1.078$ $\widehat{a} = 51.725$ $\widehat{\lambda} = 25.346$

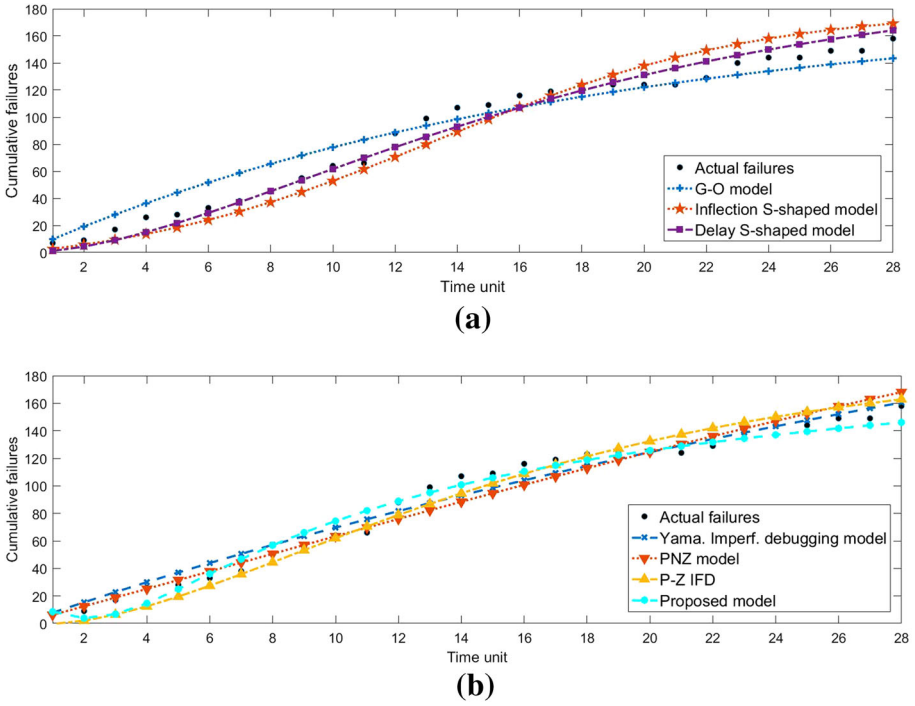

Fig. 4 Comparison of actual failure data and predicted failure data (DS2)

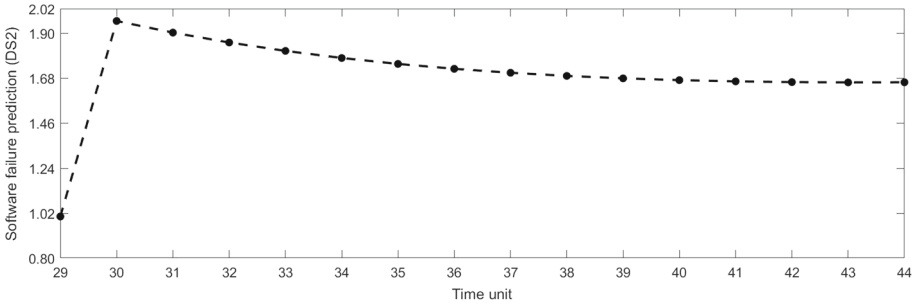

Fig. 5 Software failure prediction for time unit 29–44 (DS2)

Therefore, most software companies will release the updated version to fix the issues in the current release to increase software reliability in the operation phase and introduce some new features to improve user experience (Zhu and Pham 2017a).

5.4 Discussion of the impact of environmental factor (PoRM)

To emphasize the significance of incorporating PoRM in software reliability model, the comparison between the software reliability model with and without PoRM will be discussed in this section. Without considering PoRM and its impact, Eq. (9) will be formulated as

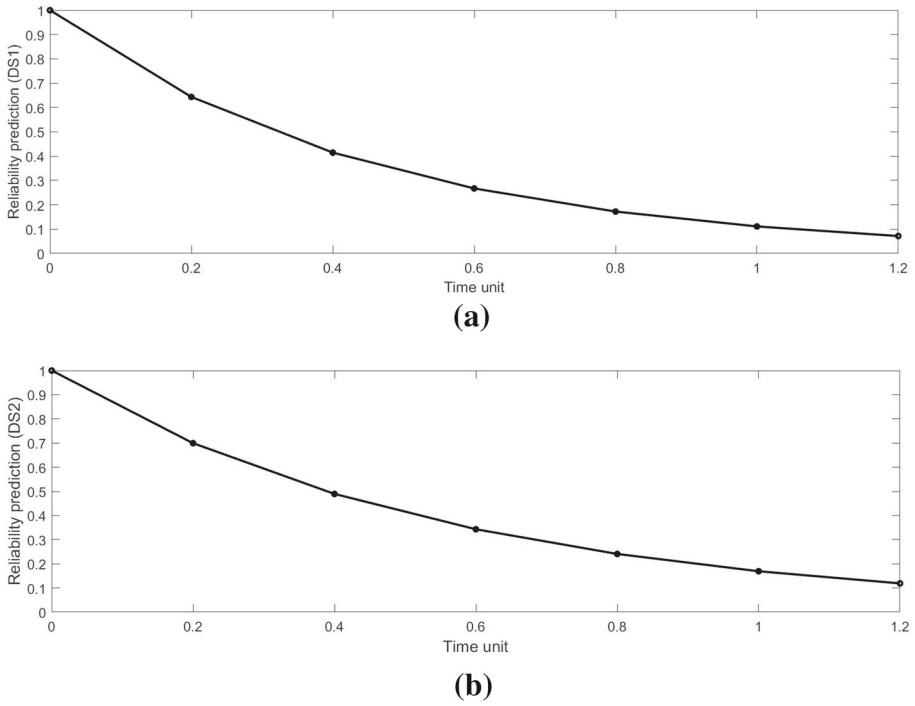

Fig. 6 Reliablity predicton for DS1 and DS2

$$\frac{d}{dt}m(t,\eta) = \left[h(t) + \dot{B}(t)\right][N(t) - m(t,\eta)] \tag{34}$$

This formulation has detailed explanation in Pham and Pham (2017). Substitute Eqs. (10), (22) and (24) into Eq. (34), the new mean value function without considering environmental factor (PoRM) can be obtained as follows

$$m(t) = \frac{1}{k}e^{kt} - \frac{e^{\frac{t}{2}}}{c+e^{bt}}\left(\frac{c+1}{k} - \frac{c}{k-\frac{1}{2}} - \frac{1}{b+k-\frac{1}{2}}\right) - \frac{ce^{kt}\left(c+e^{bt}\right)^{-1}}{k-\frac{1}{2}} - \frac{e^{(k+b)t}\left(c+e^{bt}\right)^{-1}}{b+k-\frac{1}{2}} \tag{35}$$

As an illustration for the comparison, DS1 will be used to compare the mean value function with PoRM [Eq. (25)] and the mean value function without PoRM [Eq. (35)]. The parameter estimate for the mean value function without PoRM [Eq. (35)] are $\widehat{k} = 0.109$, $\widehat{c} = 0.045$, $\widehat{b} = 0.501$. Using the criteria described in Sect. 4, we have $MSE = 103.238$, which is significantly higher than the model considering PoRM, 35.173, as shown in Table 4. We also present Fig. 7, which compares the actual failure data, the failure prediction by the model with PoRM [Eq. (25)], and the failure prediction by the model without PoRM [Eq. (34)]. We can see that the failure prediction will be more accurate with considering PoRM for OSS project data. Therefore, incorporating PoRM in software reliability model will significantly improve the predictive accuracy.

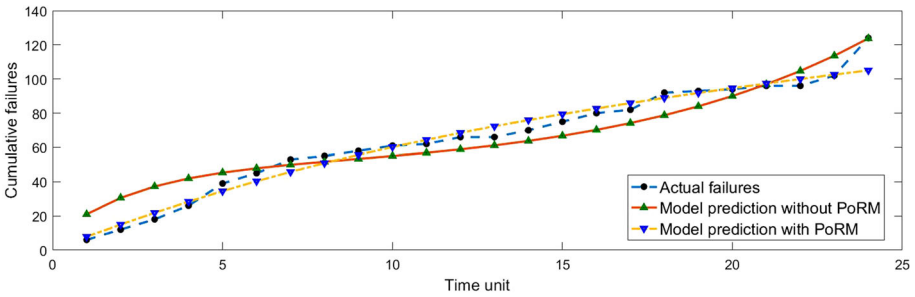

Fig. 7 Software failure prediction comparison of models with and without PoRM

6 Conclusions and future research

To the best of our knowledge, the environmental factor, in particular, *Percentage of Reused Modules* (PoRM) and the martingale framework, specifically, Brownian motion and white noise process have not been simultaneously employed in NHPP software reliability model. Lots of software developments have shifted their attention from building a new system toward composing system from the existing open source platform. On the other hand, there is an increasing trend on the adoption of software ecosystem. The development of new functionality can be occurred outside of the platform. For example, App-store styled approaches are getting popular in software community (Bosch 2010). Therefore, it is of great importance to incorporate *Percentage of Reused Modules* in the software reliability model because it brings a significant impact on open source project compared with traditional software development. Pham and Pham's (2017) recent publication concentrated on the application of martingale framework and provide a generalized solution for this type of problem, however, they did not include specific environmental factors and their distributions.

There are several research directions can be pursued in the next step. For instance, (1) we are currently investigating one environmental factor, PoRM, in this study. Two or more environmental factors can be studied in the future research. Correlation could also be considered between these environmental factors; (2) the total number of fault content is considered as a time dependent function in this study. The randomness caused by environmental factors may also affect this time dependent function; (3) the function [Eq. (21)] behavior can be further investigated.

Acknowledgements The authors would like to thank the editor and the reviewers for their valuable and constructive comments.

Appendix I

Given a generalized mean value function

$$\frac{d}{dt}m(t,\eta) = h(t,\eta)[N(t) - m(t,\eta)]$$
$$m(0) = 0$$

The general solution for the above function is

$$m(t,\eta) = e^{-\int_0^t h(s,\eta)dt} \int_0^t e^{\int_0^u h(s,\eta)ds} h(u)N(u)du = \int_0^t h(u)N(u)e^{-\int_u^t h(s,\eta)ds} du$$

Since

$$\frac{d}{du}\left(e^{-\int_u^t h(s,\eta)ds}\right) = e^{-\int_u^t h(s,\eta)ds} * h(u,\eta)$$

Thus, we have

$$\begin{aligned} m(t,\eta) &= \int_0^t N(u)d\left[e^{-\int_u^t h(s,\eta)ds}\right] \\ &= \left[N(t)e^{-\int_u^t h(s,\eta)ds}\right]_{u=t} - \left[N(t)e^{-\int_u^t h(s,\eta)ds}\right]_{u=0} - \int_0^t e^{-\int_u^t h(s,\eta)ds} d[N(u)] \\ &= N(t) - N(0)e^{-\int_0^t h(s,\eta)ds} - \int_0^t e^{-\int_u^t h(s,\eta)ds} N'(u)du \end{aligned}$$

References

Ahmadi, M., Mahdavi, I., & Garmabaki, A. H. S. (2016). Multi up-gradation reliability model for open source software. *Current trends in reliability, availability, maintainability and safety* (pp. 691–702). Berlin: Springer.

Almering, V., van Genuchten, M., Cloudt, G., & Sonnemans, P. J. (2007). Using software reliability growth models in practice. *IEEE Software, 24*(6), 82–88.

Basole, R. C., & Karla, J. (2011). On the evolution of mobile platform ecosystem structure and strategy. *Business & Information Systems Engineering, 3*(5), 313.

Bosch, J. (2010). Architecture in the age of compositionality. In *European conference on software architecture* (pp. 1–4). Springer, Berlin.

Chang, Y. C., & Liu, C. T. (2009). A generalized JM model with applications to imperfect debugging in software reliability. *Applied Mathematical Modelling, 33*(9), 3578–3588.

Clarke, P., & O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology, 54*(5), 433–447.

Cosgrove, L. (2003). *Confidence in open source growing*. CIO research report.

Fiondella, L., & Zeephongsekul, P. (2016). Trivariate Bernoulli distribution with application to software fault tolerance. *Annals of Operations Research, 244*(1), 241–255.

Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: the boundary resources model. *Information Systems Journal, 23*(2), 173–192.

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability, 28*(3), 206–211.

Harman, M., Jia, Y., & Zhang, Y. (2012). App store mining and analysis: MSR for app stores. In *2012 9th IEEE working conference on mining software repositories (MSR)* (pp. 108–111). IEEE.

Hsu, C. J., Huang, C. Y., & Chang, J. R. (2011). Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor. *Applied Mathematical Modelling, 35*(1), 506–521.

Huang, C. Y., & Kuo, S. Y. (2002). Analysis of incorporating logistic testing-effort function into software reliability modeling. *IEEE Transactions on Reliability, 51*(3), 261–270.

Huang, C. Y., Kuo, S. Y., & Lyu, M. R. (2007). An assessment of testing-effort dependent software reliability growth models. *IEEE Transactions on Reliability, 56*(2), 198–211.

Huang, C. Y., & Lo, J. H. (2006). Optimal resource allocation for cost and reliability of modular software systems in the testing phase. *Journal of Systems and Software, 79*(5), 653–664.
Huang, C. Y., & Lyu, M. R. (2005a). Optimal testing resource allocation, and sensitivity analysis in software development. *IEEE Transactions on Reliability, 54*(4), 592–603.
Huang, C. Y., & Lyu, M. R. (2005b). Optimal release time for software systems considering cost, testing-effort, and test efficiency. *IEEE Transactions on Reliability, 54*(4), 583–591.
Inoue, S., Ikeda, J., & Yamada, S. (2016). Bivariate change-point modeling for software reliability assessment with uncertainty of testing-environment factor. *Annals of Operations Research, 244*(1), 209–220.
Jeske, D. R., & Zhang, X. (2005). Some successful approaches to software reliability modeling in industry. *Journal of Systems and Software, 74*(1), 85–99.
Jin, C., & Jin, S. W. (2014). Software reliability prediction model based on support vector regression with improved estimation of distribution algorithms. *Applied Soft Computing, 15,* 113–120.
Kapur, P. K., Pham, H., Anand, S., & Yadav, K. (2011). A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. *IEEE Transactions on Reliability, 60*(1), 331–340.
Lin, C. T., & Huang, C. Y. (2008). Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. *Journal of Systems and Software, 81*(6), 1025–1038.
Lo, J. H., Huang, C. Y., Chen, Y., Kuo, S. Y., & Lyu, M. R. (2005). Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. *Journal of Systems and Software, 76*(1), 3–13.
Melo, W. L., Briand, L., & Basili, V. R. (1998). *Measuring the impact of reuse on quality and productivity in object-oriented systems*. Technical report, University of Maryland, Department of Computer Science, College Park, MD, USA.
Mikosch, T. (1998). *Elementary stochastic calculus, with finance in view* (Vol. 6). Singapore: World Scientific Publishing Co Inc.
Mörters, P., & Peres, Y. (2010). *Brownian motion* (Vol. 30). Cambridge: Cambridge University Press.
Musa, J. D. (1975). A theory of software reliability and its application. *IEEE Trans on Software Engineering, 3,* 312–327.
Özdamar, L., & Alanya, E. (2001). Uncertainty modelling in software development projects (with case study). *Annals of Operations Research, 102*(1–4), 157–178.
Pachauri, B., Dhar, J., & Kumar, A. (2015). Incorporating inflection S-shaped fault reduction factor to enhance software reliability growth. *Applied Mathematical Modelling, 39*(5), 1463–1469.
Palviainen, M., Evesti, A., & Ovaska, E. (2011). The reliability estimation, prediction and measuring of component-based software. *Journal of Systems and Software, 84*(6), 1054–1070.
Pham, H. (2007). *System software reliability*. Berlin: Springer.
Pham, H. (2014). A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. *Optimization, 63*(10), 1481–1490.
Pham, T. & Pham, H. (2017). A generalized software reliability model with stochastic fault-detection rate. *Annals of Operations Research*. https://doi.org/10.1007/s10479-017-2486-3.
Radatz, J., Geraci, A., & Katki, F. (1990). IEEE standard glossary of software engineering terminology. *IEEE Std, 610121990*(121990), 3.
Rana, R., Staron, M., Berger, C., Hansson, J., et al. (2014). Selecting software reliability growth models and improving their predictive accuracy using historical projects data. *Journal of Systems and Software, 98,* 59–78.
Sukhwani, H., Alonso, J., Trivedi, K. S., & Mcginnis, I. (2016). Software reliability analysis of NASA space flight software: A practical experience. In *2016 IEEE international conference on software quality, reliability and security* (pp. 386–397).
Teng, X., & Pham, H. (2006). A new methodology for predicting software reliability in the random field environments. *IEEE Transactions on Reliability, 55*(3), 458–468.
Wheeler, D. A. (2007). Why open source software/free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers. http://www.dwheeler.com/oss_fs_why.html.
Xie, M., & Hong, G. Y. (1998). A study of the sensitivity of software release time. *Journal of Systems and Software, 44*(2), 163–168.
Yamada, S. (2014). *Software reliability modeling: Fundamentals and applications* (Vol. 5). Tokyo: Springer.
Yamada, S., Ohtera, H., & Narihisa, H. (1986). Software reliability growth models with testing-effort. *IEEE Transactions on Reliability, 35*(1), 19–23.
Yamada, S., & Tamura, Y. (2016). *OSS reliability measurement and assessment*. Berlin: Springer.
Yang, B., & Xie, M. (2000). A study of operational and testing reliability in software reliability analysis. *Reliability Engineering & System Safety, 70*(3), 323–329.

Zachariah, B. (2015a). Optimal stopping time in software testing based on failure size approach. *Annals of Operations Research, 235*(1), 771–784.
Zachariah, B. (2015b). Optimal stopping time in software testing based on failure size approach. *Annals of Operations Research, 235*(1), 771–784.
Zhang, X., & Pham, H. (2000). An analysis of factors affecting software reliability. *Journal of Systems and Software, 50*(1), 43–56.
Zhang, X., Shin, M. Y., & Pham, H. (2001). Exploratory analysis of environmental factors for enhancing the software reliability assessment. *Journal of Systems and Software, 57*(1), 73–78.
Zhou, Y. & Davis, J. (2005). Open source software reliability model: an empirical approach. In *ACM SIGSOFT software engineering notes* (Vol. 30, No. 4, pp. 1–6). ACM.
Zhu, M., & Pham, H. (2016). A software reliability model with time-dependent fault detection and fault removal. *Vietnam Journal of Computer Science, 3*(2), 71–79.
Zhu, M. & Pham, H. (2017a). A multi-release software reliability modeling for open source software incorporating dependent fault detection process. *Annals of Operations Research*. https://doi.org/10.1007/s10479-017-2556-6
Zhu, M., & Pham, H. (2017b). Environmental factors analysis and comparison affecting software reliability in development of multi-release software. *Journal of Systems and Software, 132,* 72–84.
Zhu, M., Zhang, X., & Pham, H. (2015). A comparison analysis of environmental factors affecting software reliability. *Journal of Systems and Software, 109,* 150–160.