 doi:10.20944/preprints202207.0357.v1

Article

A Comprehensive Analysis of Proportional Intensity-based Software Reliability Models with Covariates

Siqiao Li [1,†], Tadashi Dohi [1,†*] and Hiroyuki Okamura [1,†]

[1] Hiroshima University; rel-siqiao, dohi, okamu@hiroshima-u.ac.jp
* Correspondence: dohi@hiroshima-u.ac.jp
† Current address:1-4-1 Kagamiyama, Higashi-Hiroshima City, Japan 739-8527

Abstract: This paper focuses on the so-called proportional intensity-based software reliability models (PI-SRMs), which are extensions of the common non-homogeneous Poisson process (NHPP)-based SRMs, and describe the probabilistic behavior of software fault-detection process by incorporating the time-dependent software metrics data observed in the development process. Especially we generalize the seminal PI-SRM in Rinsaka, Shibata and Dohi (2006) by introducing eleven well-known fault-detection time distributions, and investigate their goodness-of-fit and predictive performances. In numerical illustrations with four data sets collected in real software development projects, we utilize the maximum likelihood estimation to estimate model parameters with three time-dependent covariates; test execution time, failure identification work and computer time-failure identification, and examine the performances of our PI-SRMs in comparison with the existing NHPP-based SRMs without covariates. It is shown that our PI-STMs could give better goodness-of-fit and predictive performances in many cases.

Keywords: software reliability models; proportional intensity model; non-homogeneous Poisson process; time-dependent covariate; maximum likelihood estimation; goodness-of-fit performance; predictive performance

1. Introduction

Over the almost last five decades, hundreds of stochastic models, known as the *software reliability models* (SRMs), have been widely developed in the literature [1–3], to investigate and describe software fault-detection phenomena from the viewpoints of mathematical approaches. Among these models, non-homogeneous Poisson process (NHPP)-based SRMs have been identified as the most traditional but important SRMs, which have attracted extensive attention from the software reliability community in describing the stochastic behavior of the software fault-detection. The relevant investigations also validated their superiority in the goodness-of-fit performance with software fault count data compared to the other SRMs. In fact, most of the existing NHPP-based SRMs in the literature have been developed by assuming representative probability distributions to represent the time-to-fault detection, including exponential distribution [4], gamma distribution [5,6], pareto distribution [7], truncated-logistic [8] and log-logistic [9] distributions, truncated-normal distribution [10], log-normal distribution [10,11], the extreme-value distributions [12]. These NHPP-based SRMs are characterized by the mean value functions or the cumulative distribution functions (CDF) of software fault-detection time. Hence, they can qualitatively represent the typical software reliability growth phenomena and the software debugging scenarios during the software testing phase. In other words, the above approach is categorized into a black-box approach, where the software fault-detection time distribution is estimated with only the fault count data and does not depend on the knowledge/learning effects of the software product, test resources, and the process information. It should be noted that the common NHPP-based SRMs are quite simple in

 doi:10.20944/preprints202207.0357.v1

software reliability measurement and fault prediction but miss out on several software development/testing metrics data collected throughout the software development process.

As an extension of the common NHPP-based SRMs, this paper summarize the so-called proportional intensity-based software reliability models (PI-SRMs) by Rinsaka *et al.* [13], and describe the probabilistic behavior of the software fault-detection process by incorporating the time-dependent software metrics data observed in the development process. In the subsequent paper, Shibata *et al.* [14] develop a software reliability assessment tool, PI-SRAT, to automate the parameter estimation and quantify the software reliability. Especially we generalize the seminal PI-SRM in [13] by introducing several well-known fault-detection time distributions because the work in [13] limited a few kinds of software fault-detection time distributions. The advantage of PI-SRMs is to combine a regression formula to represent the dependence of software metrics data with a stochastic counting process to represent the software fault count. Similar to the well-known software reliability assessment tool in SRATS [15], we introduce eleven parametric models (baseline intensity functions) in the PI-SRM and comprehensively evaluate the potential performances, including the goodness-of-fit performance and predictive performance for the PI-SRMs. In numerical illustrations with four data sets collected in real software development projects, we apply the maximum likelihood estimation to estimate model parameters with three time-dependent covariates (software metrics); test execution time (CPU hr), failure identification work (person hr), and computer time-failure identification (CPU hr), and examine the performances of our PI-SRMs in comparison with the existing NHPP-based SRMs without covariates. Also, we investigate the dependence of software metrics in the software fault count process by checking the contribution of each software metric in the resulting regression coefficient.

The remainder of this paper is organized as follows. In Section 2, we provide a summary of the related works on PIM-based software reliability modeling approach. Section 3 summarizes the well-known NHPP-based SRMs and the maximum likelihood estimation for estimating model parameters. In Section 4, we give the definition of PI-SRMs and introduce the piecewise continuous mean value function to represent the cumulative and non-cumulative software development/test effect. Section 5 focuses on numerical experiments to facilitate our analysis effects of respective software metrics on the software fault count process. Finally, the paper is concluded in Section 6.

2. Related Works

It has been known that there was a strong correlation between software quality and some kinds of software development metrics. McCabe [16] presented various software engineering metrics that serve as units of measurement for the development of human resources, software products, and processes. Halstead [17] also emphasized the importance of software science and developed deterministic equations for estimating the quantity of residual faults in software as a function of programming effort. Putnam [18] and Takahashi and Kamayachi [19] demonstrated empirical relationships between the software product development environmental factors (programming language, coding techniques, reusability of existing code, programmer skill, *etc.*), and software fault characteristics. With numerous metrics data, Pillai and Nair [20] described an estimation problem of software cost and development effort. The methodologies mentioned above, on the other hand, are basically deterministic and cannot account for the uncertainty of software fault-detection mechanisms in the testing.

In general, it is known that software metrics can be divided into four categories; product metrics, development/test metrics, deployment/usage metrics, and software-hardware configurations metrics. So, when both the software metrics and fault-detection data are utilized in the software reliability analysis, it can be expected that the assessment of the software reliability accurately will be more plausible and can be improved. Khoshgoftaar and Munson [21] and Khoshgoftaar *et al.* [22–24] introduced linear and non-linear regressions into the predictive modeling approaches to quantify the software quality with several

 doi:10.20944/preprints202207.0357.v1

software complexity metrics data. Schnieidewind [25,26] also used regression models to quantify the software maintenance process. For the prediction of the field defect-occurrence rate, Li *et al.* [27] considered a defect-occurrence projection (metrics-based) method based on the exponential smoothing and classical moving average. Khoshgoftaar *et al.* [28] used the pure and the zero-inflated Poisson regression approaches to develop software fault prediction models to predict the rank-order of software modules. Amasaki *et al.* [29] utilized the rank correlation coefficient and the logistic regression model to identify the fault data trend and evaluate the software quality, which is quantified by the number of detected faults after shipping. Unfortunately, these models [21–24,27–29] fail to deal with both the time series metrics data and the regression data simultaneously.

Ascher [30,31], Bendell [32], Evanco and Lacovara [33], Evanco [34] and Nishio and Dohi [35] used the proportional hazard model (PHM) or Cox regression model [36] to integrate the software development metrics and/or environmental factors and defined the software fault-detection time distribution by taking the time-series metrics data into account as the covariate [37,38]. Pham [3] investigated a dynamic variant of PHM and constructed an enhanced PHM based on a continuous-time Markov chain. However, since the covariates representing the development effort were composed of 0-1 binary values in their modeling, the resulting PHM-based modeling approach lost its validation if the cumulative effect of software development/testing effort reported in [39] was analyzed. By observing the behavior of myopic software debugging phenomena, Shibata *et al.* [40] introduced the discrete-time PHM into the cumulative Bernoulli trial process and proposed a software metrics-based modeling approach. Okamura *et al.* [41] assumed the logistic regression instead of the Cox regression and proposed a different discrete-time multi-factor modeling framework. Kuwa and Dohi [42,43] further extended the metrics-based SRMs with the logistic and Cox regressions to improve both goodness-of-fit and predictive performances. Nagaraju *et al.* [44] applied the discrete-time Cox-regression-based SRMs to an optimal test activity allocation problem.

It is worth mentioning that the discrete-time metrics-based SRMs in [40–43] were quite convinced in modeling because both the logistic and Cox regression approaches were consistently taken into consideration. However, the discrete-time metrics-based SRMs depend on the discrete fault-detection time distribution. In general, it is known that handling the continuous probability distributions is much easier than the discrete probability distributions. In fact, in the references [40–43], the authors dealt with very few discrete probability distributions, such as the geometric distribution, negative binomial distribution, and discrete Weibull distribution. On one hand, the PI-SRMs in [13] are based on the continuous probability distribution and can represent many software fault-detection patterns in the framework. The basic idea of PI-SRMs has come from the proportional intensity model (PIM) by Lawless [45], which is an extension of the common PHM in terms of time series analysis. However, it is worth noting that the PIM in [45] is not directly applicable to the non-decreasing cumulative data such as the software fault count. Hence, Rinsaka *et al.* [13] proposed a modified NHPP-based SRM with piecewise continuous mean value function with monotone increasing property to apply the PIM to the software fault count data analysis.

3. NHPP-Based Software Reliability Modeling

From the viewpoint of modeling and statistical estimation, suppose the stochastic counting process with the following four conditions:

(i) $N(0) = 0$
(ii) $\{N(t), t \geq 0\}$ has independent increment
(iii) $\Pr\{N(t + \Delta t) - N(t) \geq 2\} = o(\Delta t)$
(iv) $\Pr\{N(t + \Delta t) - N(t) = 1\} = \lambda(t)\Delta t + o(\Delta t)$,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

where $N(t)$ denotes the total number of events that occurred up to and including time t, $\lambda(t)$ is a continuous (deterministic) function of time t, called the intensity function, and $o(\Delta t)$ is the higher-order term of infinitesimal time Δt satisfying

$$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0. \tag{1}$$

Then the stochastic counting process $\{N(t), t \geq 0\}$ can be regarded as the non-homogeneous Poisson process (NHPP). Since NHPP is a typical Markov process with a time-dependent transition rate, the probability mass function (PMF) of NHPP is given by

$$\Pr\{N(t) = n\} = \frac{\{H(t)\}^n}{n!} e^{-H(t)}, \tag{2}$$

where

$$H(t) = \int_0^t \lambda(x) dx = \mathrm{E}[N(t) = n] \tag{3}$$

is the mean value function of NHPP with $H(0) = 0$. It denotes the expected cumulative number of software faults detected by time t.

In software reliability engineering, two commonly used modeling assumptions are made:

(i) Software faults are detected at independent and identically distributed (i.i.d.) random times with the non-degenerate cumulative distribution function (CDF), $F(t; \boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ is a free parameter vector.

(ii) The total number of software faults remaining in software before testing, say, at time $t = 0$, is a Poisson random variable with parameter ω (> 0).

Under the above two assumptions, it can be confirmed that the software fault-detection process $N(t)$ follows an NHPP with mean value function $H(t; \boldsymbol{\theta}) = \omega F(t; \boldsymbol{\alpha})$, where $\boldsymbol{\theta} = (\omega, \boldsymbol{\alpha})$ and $\lim_{t \to \infty} H(t; \boldsymbol{\theta}) = \omega$ (> 0). The resulting bounded mean value function implies that the initial number of residual faults in software is finite.

The key idea in the traditional software reliability modeling was to determine the mean value function $H(t; \boldsymbol{\theta})$ or the fault-detection time CDF $F(t; \boldsymbol{\alpha})$, to fit the software fault count data. In the software reliability assessment tool on the spreadsheet (SRATS), Okamura and Dohi [15] implemented the NHPP-based SRM with eleven representative software fault-detection time CDFs belong to the generalized exponential distribution family and the extreme-value distribution family. All of them have been developed in the heavily cited references [4–12]. We apply these eleven NHPP-based SRMs to our PI-SRMs. More specifically, Table 1 presents the eleven CDFs, ~~their~~ associated abbreviations and intensity functions.

Before closing this section, we summarize the maximum likelihood estimation for the common NHPP-based SRMs. Suppose that the cumulative number of software faults detected by each testing time t_k $(k = 1, 2, \ldots, n)$, measured in calendar time, is denoted by y_k. For the time interval (group) data (t_k, y_k) $(k = 1, 2, \ldots, n)$, the likelihood function and log-likelihood function with unknown parameter $\boldsymbol{\theta}$ are given by

$$L(\boldsymbol{\theta}) = \exp[-H(t_n; \boldsymbol{\theta})] \prod_{k=1}^{n} \frac{\{H(t_k; \boldsymbol{\theta}) - H(t_{k-1}; \boldsymbol{\theta})\}^{y_k - y_{k-1}}}{(y_k - y_{k-1})!} \tag{4}$$

and

$$\begin{aligned} \mathrm{LLF}(\boldsymbol{\theta}) &= \sum_{k=1}^{n} \ln[H(t_k; \boldsymbol{\theta}) - H_t(t_{k-1}; \boldsymbol{\theta})](y_k - y_{k-1}) - H(t_n; \boldsymbol{\theta}) \\ &\quad - \sum_{k=1}^{n} \ln[(y_k - y_{k-1})!], \end{aligned} \tag{5}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Table 1. The existing NHPP-based SRMs.

Models	$\lambda(t;\boldsymbol{\theta})$ & $F(t;\boldsymbol{\alpha})$
Exponential distribution (exp) [4]	$\lambda(t;\boldsymbol{\theta}) = \omega b e^{-bt}$ $F(t;\boldsymbol{\alpha}) = 1 - e^{-bt}$
Gamma distribution (gamma) [5],[6]	$\lambda(t;\boldsymbol{\theta}) = \omega \frac{e^{-\frac{t}{c}}\left(\frac{t}{c}\right)^{b-1}}{c\Gamma(b)}$ $F(t;\boldsymbol{\alpha}) = \int_0^t \frac{c^b s^{b-1} e^{-cs}}{\Gamma(b)} ds$
Pareto distribution (pareto) [7]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega bc\left(\frac{c}{c+t}\right)^{b-1}}{(c+t)^2}$ $F(t;\boldsymbol{\alpha}) = 1 - (\frac{b}{t+b})^c$
Truncated normal distribution (tnorm) [10]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{(c-t)^2}{2b^2}}}{\sqrt{2\pi}b\left(1-\frac{1}{2}\mathrm{erfc}\left(\frac{c}{\sqrt{2}b}\right)\right)}$ $F(t;\boldsymbol{\alpha}) = \frac{1}{\sqrt{2\pi}b}\int_{-\infty}^{t} e^{-\frac{(s-c)^2}{2b^2}} ds$
Log-normal distribution (lnorm) [11],[10]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{(c-\log(t))^2}{2b^2}}}{\sqrt{2\pi}btt}$ $F(t;\boldsymbol{\alpha}) = \frac{1}{\sqrt{2\pi}b}\int_{-\infty}^{t} e^{-\frac{(s-c)^2}{2b^2}} ds$
Truncated logistic distribution (tlogist) [8]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{t-c}{b}}}{b\left(1-\frac{1}{e^{c/b}+1}\right)\left(e^{-\frac{t-c}{b}}+1\right)^2}$ $F(t;\boldsymbol{\alpha}) = \frac{1-e^{-bt}}{1+ce^{-bt}}$
Log-logistic distribution (llogist) [9]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{\log(t)-c}{b}}}{bt\left(e^{-\frac{\log(t)-c}{b}}+1\right)^2}$ $F(t;\boldsymbol{\alpha}) = \frac{(bt)^c}{1+(bt)^c}$
Truncated extreme-value maximum distribution (txvmax) [12]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{t-c}{b}-e^{-\frac{t-c}{b}}}}{b\left(1-e^{-e^{c/b}}\right)}$ $F(t;\boldsymbol{\alpha}) = e^{-e^{-\frac{t-c}{b}}}$
Log-extreme-value max maximum distribution (lxvmax) [12]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega c e^{-\left(\frac{t}{b}\right)^{-c}}\left(\frac{t}{b}\right)^{-c-1}}{b}$ $F(t;\boldsymbol{\alpha}) = e^{-\left(\frac{t}{b}\right)^{-c}}$
Truncated extreme-value minimum distribution (txvmin) [12]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{-c-t}{b}-e^{-\frac{-c-t}{b}}+e^{c/b}}}{b}$ $F(t;\boldsymbol{\alpha}) = e^{-e^{-\frac{t-c}{b}}}$
Log-extreme-value minimum distribution (lxvmin) [46]	$\lambda(t;\boldsymbol{\theta}) = \frac{\omega e^{-\frac{-c-\log(t)}{b}-e^{-\frac{-c-\log(t)}{b}}}}{bt}$ $F(t;\boldsymbol{\alpha}) = e^{-e^{-\frac{t-c}{b}}}$
	$(\omega > 0, b > 0, c > 0)$

respectively, where $(t_0, y_0) = (0,0)$. Note that almost all software fault count data observed in practice are the group data, because the software debugging is often made in the distributed testing environment, and that measurement of execution time to detect each software fault, which is measured by CPU time, is almost impossible in industry. Finally, we can obtain the maximum likelihood (ML) estimate $\hat{\boldsymbol{\theta}}$ by maximizing Equation 5 with respect to $\boldsymbol{\theta}$.

4. Proportional Intensity Model

4.1. Model Description

In this section, we introduce a PI-SRM compatible with the maximum likelihood estimation and incorporate several testing-effort factors observed on respective testing dates. Suppose that l types of software metrics data, $\mathbf{x}_k = (x_{k1}, \cdots, x_{kl})$ $(k = 1, 2, \ldots, n)$, are observed at each testing time t_k $(= 0, 1, 2, \ldots, n)$. For the analytical purpose, we assume that each software metric $\mathbf{x}_k$ is dependent on the cumulative testing time t_k, and can be considered as a time-dependent function, denoted by $\mathbf{x}_k(t_k)$. In fact, this sort of parameter is referred to as a time-dependent covariate [37,38] in statistics and has been

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

widely investigated in the context of the Cox regression-based proportional hazard model (PHM). Define the intensity function for our PI-SRM by

$$\lambda_x(t_k, \mathbf{x}_k; \boldsymbol{\theta}, \boldsymbol{\beta}) = \lambda_0(t_k; \boldsymbol{\theta}) g(\mathbf{x}_k; \boldsymbol{\beta}), \tag{6}$$

with the regression coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_l)$ and the baseline intensity $\lambda_0(t_k; \boldsymbol{\theta})$ (> 0), and the covariate function $g(\mathbf{x}_k; \boldsymbol{\beta})$ (> 0). When $g(\mathbf{x}_k; \boldsymbol{\beta}) = 1$ for any $\mathbf{x}_k$, the PI-SRMs are reduced to the NHPP-based SRMs with the baseline intensity $\lambda_0(t; \boldsymbol{\theta})$. Based on the idea of common Cox regression PHM, it is appropriate to assume the following exponential form for the covariate function:

$$g(\mathbf{x}_k; \boldsymbol{\beta}) = \exp(\mathbf{x}_k \boldsymbol{\beta}). \tag{7}$$

In the literature [36–38], the above form is widely accepted to make the analysis easy and flexible. Lawless [45] also analyzed the event count data in actual medical applications with the same exponential covariate function. Note that the time-independent covariates considered by Lawless [45] were the binary data taking 0 and 1. Rinsaka *et al.* [13] proposed an intuitive but reasonable model to deal with the effect of the cumulative number of software faults and the software metrics in the covariate function. Define the mean value function for the given data $(t_k, y_k, \mathbf{x}_k)$ $(k = 1, 2, \ldots, n)$ by

$$\begin{aligned}
H_p(t_1; \boldsymbol{\theta}, \boldsymbol{\beta}) &= \int_0^{t_1} \lambda_0(u; \boldsymbol{\theta}) \exp(\mathbf{x}_1 \boldsymbol{\beta}) du, && (8)\\
H_p(t_2; \boldsymbol{\theta}, \boldsymbol{\beta}) &= \int_{t_1}^{t_2} \lambda_0(u; \boldsymbol{\theta}) \exp(\mathbf{x}_2 \boldsymbol{\beta}) du + H_p(t_1; \boldsymbol{\theta}, \boldsymbol{\beta}), && (9)\\
&\vdots \\
H_p(t_k; \boldsymbol{\theta}, \boldsymbol{\beta}) &= \sum_{i=1}^{k} \exp(\mathbf{x}_i \boldsymbol{\beta}) \int_{t_{i-1}}^{t_i} \lambda_0(u; \boldsymbol{\theta}) du \\
&= \sum_{i=1}^{k} \exp(\mathbf{x}_i \boldsymbol{\beta}) \times [H_0(t_i; \boldsymbol{\theta}) - H_0(t_{i-1}; \boldsymbol{\theta})], && (10)
\end{aligned}$$

where $H_0(t_i; \boldsymbol{\theta}) = \int_0^{t_i} \lambda_0(u; \boldsymbol{\theta}) du$. It is seen again that the PI-SRM can be reduced to the common NHPP-based SRM when $\beta_j = 0$ for all j $(= 1, 2, \cdots, l)$. By introducing $H_p(t; \boldsymbol{\theta}, \boldsymbol{\beta})$, we confirm that the monotone property of the mean value function with respect to testing time r can be guaranteed. Substituting the intensity function in Table 1 in to the baseline intensity $\lambda_0(t; \boldsymbol{\theta})$, we obtain the eleven PI-SRMs corresponding to the NHPP-based SRMs in SRATS [15].

4.2. Maximum likelihood estimation

We also utilize the maximum likelihood estimation to estimate the parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ of PI-SRM. For the fault count data (t_k, y_k) and software metrics data $\mathbf{x}_k = (x_{k1}, \cdots, x_{kl})$ $(k = 1, 2, \ldots, n)$, we define the likelihood function by

$$L(\boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{k=1}^{n} \frac{\{H_p(t_k; \boldsymbol{\theta}, \boldsymbol{\beta}) - H_p(t_{k-1}; \boldsymbol{\theta}, \boldsymbol{\beta})\}^{y_k - y_{k-1}}}{(y_k - y_{k-1})!} \exp[-H_p(t_n; \boldsymbol{\theta}, \boldsymbol{\beta})], \tag{11}$$

so that the log-likelihood function of PI-SRM can be written as

$$\begin{aligned}
\mathrm{LLF}(\boldsymbol{\theta}, \boldsymbol{\beta}) &= \sum_{k=1}^{n} \ln\big[H_p(t_k; \boldsymbol{\theta}, \boldsymbol{\beta}) - H_p(t_{k-1}; \boldsymbol{\theta}, \boldsymbol{\beta})\big](y_k - y_{k-1}) \\
&\quad - \sum_{k=1}^{n} \ln[(y_k - y_{k-1})!] - H_p(t_n; \boldsymbol{\theta}, \boldsymbol{\beta}). && (12)
\end{aligned}$$

By maximizing Equation 12 with Newton-Raphson method, we obtain the maximum likelihood estimates $(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}})$ of PI-SRM.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

5. Numerical Examples

In our numerical examples, four software fault count data with software metrics are used, where these data are measured in the real-time command and control system development projects [39]. Details are shown in Table 2, in which three software metrics data; failure identification work, execution time, computer time-failure identification, are involved in addition to the cumulative number of software faults detected at each testing time (calendar week in [39]). We quantitatively evaluate the goodness-of-fit performances of eleven PI-SRMs and evaluate the predictive performances via the above four time-dependent metrics data as the covariates. In the following discussion, we consider two patterns in dealing with software metrics. One is to input the software metrics as the cumulative $\mathbf{x}_k = (x_{k1}, \cdots, x_{kl})$, the other the difference $\mathbf{x}_k = \left(x_{k1} - x_{(k-1)1}, \cdots, x_{kl} - x_{(k-1)l}\right)$, where l is the number of time-dependent metrics data in each data set and $k = 0, 1, 2, \cdots, n$. The main concern here is to investigate the effects of cumulative values of software metrics on the contribution to the software fault count. For instance, we examine the difference between the cumulative length of test execution time by the present testing time and the test execution time spend at the same testing time.

Table 2. Data sets.

Data	No. faults	Testing days
GDS1	136	21
GDS2	54	17
GDS3	38	14
GDS4	53	16
Metrics Data:	Failure identification work, Execution time, Computer time-failure identification.	

5.1. Goodness-of-fit Performance

For our PI-SRMs, we assume eleven baseline intensity functions in Table 1 and compare them to investigate the effects of each time-dependent software metric data on the stochastic behavior of the cumulative number of software faults detected in the testing phase. We calculate the maximum likelihood estimates $(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}})$ of covariate $g(\mathbf{x}_k; \boldsymbol{\beta}) = \exp(\mathbf{x}_k \boldsymbol{\beta})$ for all combinations of software metrics data in Table 2 and consider a total of 7 combinations, as shown in Table 3. By deriving the corresponding log-likelihood (LLF), the Akaike information criterion (AIC) and mean squared error (MSE) are used to evaluate the goodness-of-fit performances of our PI-SRMs, where

$$\text{AIC} = -2\text{LLF}(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}}) + 2\pi(\text{number of free parameters}), \tag{13}$$

and

$$\text{MSE} = \frac{1}{n}\sqrt{\sum_{k=1}^{n}(y_k - H_p(t_k; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}}))^2}. \tag{14}$$

The lower the AIC/MSE, the better SRM in terms of goodness-of-fit to the fault count data.

In Figure 1, we plot the cumulative number of detected software faults in GDS1 and the estimated mean value functions in the best-fitted SRMs, where we select the best model with the minimum AIC for the common NHPP-based SRMs without software metrics (orange curve) in SRATS [15], PI-SRM with cumulative software metrics (red curve), and PI-SRM with non-cumulative software metrics (blue curve), among eleven intensity functions. At first glance, it can be seen that the three curves exhibit almost similar behavior, but a closer look reveals that our PIMs can show more complex behaviors than existing NHPP-based SRM without software metrics. Figure 2 illustrates the behavior of the estimated number of detected fault counts at each testing time interval in GDS1, where the same models as Figure 1 are used for comparison, and the orange bar-chart represents the actual number of software faults in each testing week. The result explains that our two PI-SRMs

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Table 3. Combination of covariates $g(\mathbf{x}_{kl};\boldsymbol{\beta})$.

	$g(\mathbf{x}_{kl};\boldsymbol{\beta})(l=1,2,3)$
Combination I	$\exp(\beta_0+x_{k1}\beta_1)$
Combination II	$\exp(\beta_0+x_{k2}\beta_2)$
Combination III	$\exp(\beta_0+x_{k3}\beta_3)$
Combination IV	$\exp(\beta_0+x_{k1}\beta_1+x_{k2}\beta_2)$
Combination V	$\exp(\beta_0+x_{k1}\beta_1+x_{k3}\beta_3)$
Combination VI	$\exp(\beta_0+x_{k2}\beta_2+x_{k3}\beta_3)$
Combination VII	$\exp(\beta_0+x_{k1}\beta_1+x_{k2}\beta_2+x_{k3}\beta_3)$
x_{k1} : Execution time, x_{k2} : Failure identification work.	
x_{k3} : Computer time-failure identification.	

could show the more better goodness-of-fit performances than the existing NHPP-based SRM without software metrics and could catch up with the detailed trend on the software fault count.

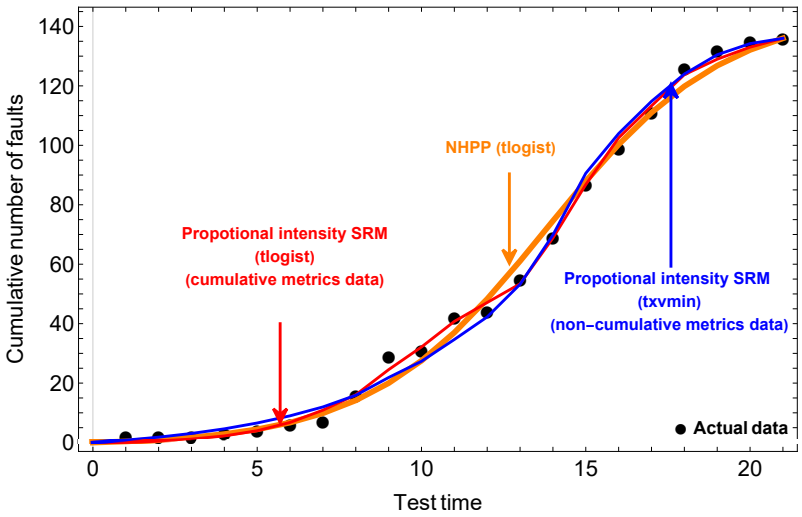

Figure 1. Behavior of estimated cumulative number of software faults in GDS1.

To compare our PI-SRMs with the common NHPP-based SRMs without software metrics more precisely, we present the best AIC results for four time-dependent metrics data in Table 4. By comparing our two PI-SRMs with cumulative/non-cumulative metrics values, we investigate how to deal with the software metrics data in software fault data analysis. From the results in Table 4, it is found that our PI-SRMs are more appealing in software reliability modeling and outperform the existing NHPP-based SRMs without software metrics in terms of goodness-of-fit. In the comparison of two patterns with cumulative/non-cumulative metric data, it is seen that the non-cumulative software metrics tend to show the better fitting results except in GDS4. Note that the difference of AIC between cumulative/non-cumulative metric patterns is minimal and negligible. So, our conclusion on the goodness-of-fit performance is that the PI-SRM with non-cumulative software metric data should be better. Also, in Table 4, it is observed that both the execution time and failure identification work could contribute to the goodness-of-fit performance in the PI-SRMs. Hence, the measurement of test execution time and failure identification work can help understand the software fault count in the testing phase more accurately and is useful to monitor the software testing progress.

5.2. Predictive Performance

Next, we are concern about investigating the predictive performances of our PI-SRMs. In each observation point n' $(1 \leq n' < n)$ when 50% or 80% of the whole data are available, we predict the future behavior of the cumulative number of software faults. To assess

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

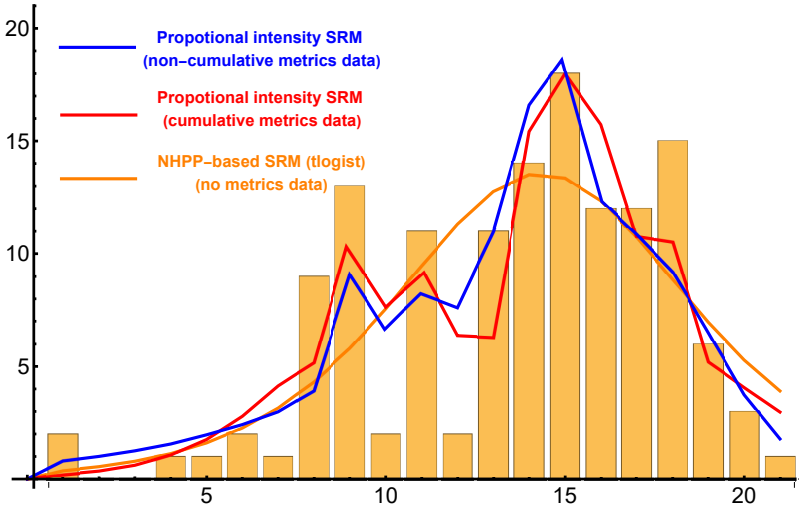

Figure 2. Behavior of estimated number of software faults in each time interval in GDS1.

Table 4. Goodness-of-fit performance based on AIC.

	Model	AIC	MSE	$\hat{\boldsymbol{\beta}}$
(i) Best proportional intensity model (cumulative metrics data)				
GDS1	tlogist-VI	110.114	**0.470**	$\hat{\beta_0} = -2.5903, \hat{\beta_2} = -0.0805, \hat{\beta_3} = 0.0277$
GDS2	tlogist-III	69.785	0.282	$\hat{\beta_0} = 1.7326, \hat{\beta_3} = 0.1406$
GDS3	txvmin-II	57.281	0.289	$\hat{\beta_0} = -3.7048, \hat{\beta_2} = 1.2197$
GDS4	exp-I	**81.059**	0.612	$\hat{\beta_0} = 4.6132, \hat{\beta_1} = -0.1659$
(ii) Best proportional intensity model (non-cumulative metrics data)				
GDS1	txvmin-II	**109.015**	0.721	$\hat{\beta_0} = 2.9503, \hat{\beta_2} = 0.0206$
GDS2	llogist-II	**67.352**	**0.261**	$\hat{\beta_0} = -0.4155, \hat{\beta_2} = 0.0447$
GDS3	gamma-II	**50.696**	**0.221**	$\hat{\beta_0} = 0.6061, \hat{\beta_2} = 1.1493$
GDS4	exp-VI	81.131	**0.450**	$\hat{\beta_0} = 3.8840, \hat{\beta_2} = -0.2963, \hat{\beta_3} = 0.8060$
(iii) Best SRATS (no metrics data)				
GDS1	tlogist	116.891	0.820	-
GDS2	llogist	73.053	0.501	-
GDS3	lxvmax	61.694	0.481	-
GDS4	txvmin	79.761	0.530	-

the predictive ability, we apply the prediction squared error (PMSE) as the predictive performance measure, where

$$\mathrm{PMSE} = \frac{1}{n-\acute{n}} \sqrt{\sum_{k=\acute{n}+1}^{n} \left[y_k - H_p(t_k; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}}) \right]^2}. \tag{15}$$

The smaller the PMSE, the better the prediction performance of the model. As expected easily, when we predict the number of software faults detected in the future, both the software metrics $\mathbf{x}_k$ $(k = 1, 2, \ldots, n)$ as well as the regression coefficient $\boldsymbol{\beta}$ must be estimated. The regression coefficients are available by applying the plug-in estimates (maximum likelihood estimates) with the past observation. However, the difficulty when the PI-SRMs are used arises since we have to predict the software metrics themselves in the future. In our numerical experiments, we consider the following three cases:

Case I: All the test/development metrics data are completely known through the testing phase in advance, so the software testing expenditures are exactly given in the testing.

Case II: The test/development metrics data do not change from the observation point in the future.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Case III: The test/development metrics data experienced in the future are regarded as independent random variables and predictable by any statistical method.

Case I corresponds to the case where the software test plan is established and there is no confusion in the software testing phase. Case II implicitly assumes that the observation point is regarded as the release point of software because no testing effort will be spent in the operational phase. Case III would be the most plausible case in software testing. In this case, we are requested to introduce any statistical model to investigate the test/development metrics data. We employ two elementary regression methods; linear regression and exponential regression, to predict the future software metrics data. More specifically, we assume that the metric data $\mathbf{x}_k$ before the observation point n' and the corresponding time point t_k have been observed with $k = 1, 2, \ldots, n'$. Next, our goal is to calculate the predictive value of the metric data $\hat{\mathbf{x}}_k$ between a given time period $(t_{n'+1},\ t_n)$, by introducing the independent variable $T = \{t_{n'+1}, t_{n'+2}, \ldots t_n\}$ into the linear regression equation

$$\hat{\mathbf{x}}_k = \delta_1 + \delta_2 t_k \tag{16}$$

with intercept δ_1 and coefficient δ_2 can derived by

$$\delta_1 = \frac{\left(\sum_{k=1}^{n'} \mathbf{x}_k\right)\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)\left(\sum_{k=1}^{n'} t_k \mathbf{x}_k\right)}{n'\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)^2} \tag{17}$$

and

$$\delta_2 = \frac{n'\left(\sum_{k=1}^{n'} t_k \mathbf{x}_k\right) - \left(\sum_{k=1}^{n'} t_k\right)\left(\sum_{k=1}^{n'} \mathbf{x}_k\right)}{n'\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)^2} \tag{18}$$

respectively. Similar to the linear regression method, we can also obtain the predictive values of the metric data $\hat{\mathbf{x}}_k$ by importing variable $T = \{t_{n'+1}, t_{n'+2}, \ldots t_n\}$ into the exponential regression equation

$$\hat{\mathbf{x}}_k = \delta_3 {\delta_4}^{t_k}, \tag{19}$$

where the correlation coefficients δ_3 and δ_4 are given by

$$\delta_3 = \exp\left(\frac{\left(\sum_{k=1}^{n'} \ln \mathbf{x}_k\right)\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)\left(\sum_{k=1}^{n'} t_k \ln \mathbf{x}_k\right)}{n'\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)^2}\right) \tag{20}$$

and

$$\delta_4 = \exp\left(\frac{n'\left(\sum_{k=1}^{n'} t_k \ln \mathbf{x}_k\right) - \left(\sum_{k=1}^{n'} t_k\right)\left(\sum_{k=1}^{n'} \ln \mathbf{x}_k\right)}{n'\left(\sum_{k=1}^{n'} {t_k}^2\right) - \left(\sum_{k=1}^{n'} t_k\right)^2}\right) \tag{21}$$

respectively. Note that with Equation 20 and Equation 21, it can be easily found that the exponential regression is not appropriate for making the prediction when the non-cumulative metric data in PI-SRMs are used, because the variable may take 0, and the correlation coefficient may not be calculated theoretically. So, we totally consider seven patterns of estimated development/test metrics data in the future phase in the above three cases, and investigate the predictive performances of our PI-SRMs.

Figures 3 and 4 depict the prediction results of the cumulative number of software faults in GDS1 at 50% observation and 80% observation, respectively. It is not difficult to find that our two PI-SRMs could show a completely different predictive trend than common NHPP-based SRMs. However, we can recognize that the closer increasing trend to the underlying software fault count data, no matter the prediction length is long or short, especially, in the testing phase after 50% and 80% observation points. The quantitative comparison in terms of predictive performance is investigated in Tables 5 and 6, where

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

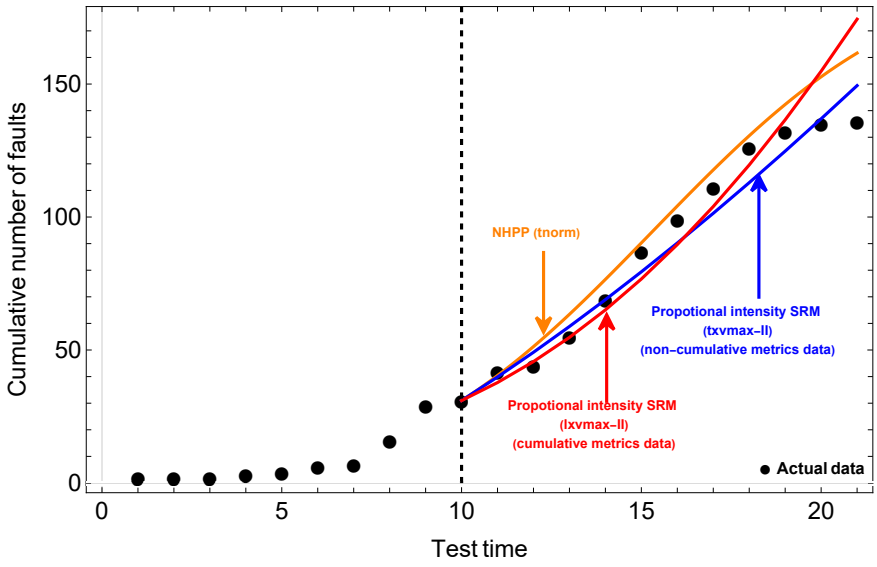

Figure 3. Behavior of the predicted cumulative number of software faults with PI-SRMs and common NHPP-based SRM in GDS1 (50% observation point).

we present the PMSE in four data sets at 50 % observation point and 80 % observation point, respectively. Here we select the best SRMs with the smallest PMSE in PI-SRMs with cumulative/non-cumulative software metric data in CASE I, CASE II, and CASE III, and the existing NHPP-based SRMs. In these tables, the bold font marks the best SRMs with minimum PMSE in each data set. From these results, it is immediate to see that our PI-SRMs could still outperform the existing NHPP-based SRMs in all the data sets. We can also find that utilizing the estimated metrics data in Case II, i.e., when the test/development metrics data do not change in the future tends to give the more better predictive performances than the other two cases in many cases (GDS1 50%, GDS2 50%, GDS3 50%, GDS4 50%, and GDS4 80%). So in 5 out of 8 (GDS2 50%, GDS4 50%, GDS2 80%, GDS3 80%, and GDS4 80%), our PI-SRMs with non-cumulative metric data could provide the minimum PMSE. More specifically, Combination II of software metrics in Table 3 could give the minimum PMSEs in GDS1 80%, GDS3 80%, and GDS4 80% data sets with non-cumulative software metric data and GDS1 50%, GDS2 50% with cumulative software metric data, respectively. The remaining three minimum PMSEs were given in the PI-SRMs with Combinations V, VI, and VII in Table 3. Finally, by carefully checking the prediction results in Tables 4 and Tables 5, we can conclude that the failure identification work is the most important development metric in prediction and leads to improving the software fault prediction accurately.

5.3. Software Reliability Assessment

In the previous argument, we have confirmed that our PI-SRMs could show the better predictive performances than the existing NHPP-SRMs in all cases. In the next step, we wish to quantify the software reliability, which is defined as the probability that the software after release is fault-free. Let $R(t_l \mid t_m) = Pr\{N(t_m) - N(t_l) = 0 \mid N(t_l) = n\}$ denote the software reliability in the operational phase $(t_l, t_m]$, where t_l is the release point. Then, from the NHPP assumption, it is easy to obtain

$$R(t_l \mid t_m) = \exp\Big[H_p(t_m; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}}) - H_p(t_l; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\beta}})\Big]. \tag{22}$$

In our numerical example, we set $t_m = 2t_l$, say, the operational period is twice the length, and assume that the software metrics $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3})$ are constant in the time interval (t_l, t_m), since the software product has not been tested after the release time t_l. We assess the software reliability quantitatively with the best PI- SRMs, which are selected with the minimum AIC at the release time point $t_l = t_n$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

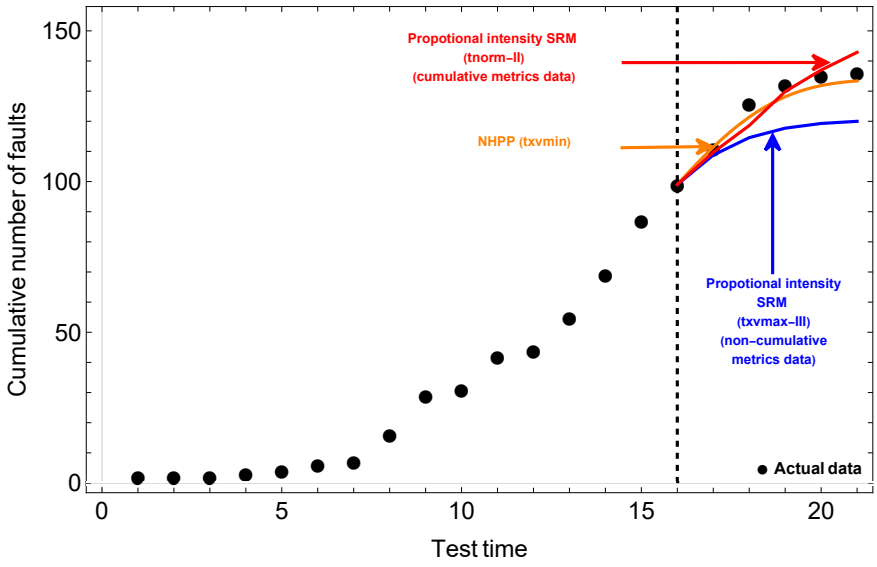

Figure 4. Behavior of the predicted cumulative number of software faults with PI-SRMs and common NHPP-based SRM in GDS1 (80% observation point).

Table 7 presents the comparison results of our PI-SRMs with the existing NHPP-based SRMs. It can be seen that our PI-SRMs with cumulative/non-cumulative software metrics could provide the larger software reliability than the common NHPP-based SRMs without software metrics. This result implies that if the PI-SRMs are reliable in goodness-of-fit and predictive performances, they are more inclined to provide positive decisions in terms of software reliability assessment, and the NHPP-based SRMs without software metrics tend to underestimate the software reliability. On the other hand, we also note that in all four data sets, the software reliability estimated by almost all of the SRMs, except in txvmin-II PI-SRM in GDS3 and txvmin NHPP-based SRM in GDS4, are not promising. This observation also implies that in time period $t_m - t_l$, these SRMs tend to give the false alarms from the viewpoint of safety, so that the software products under testing seem to require more tests to meet the software reliability requirement.

6. Conclusions

This paper presented the proportional intensity NHPP-based SRMs (PI-SRMs in short) with eleven representative baseline intensity functions, which could incorporate multiple time-dependent cumulative/non-cumulative software development/test metrics data. In our numerical experiments with actual software project data, we have quantitatively evaluated the goodness-of-fit and predictive performances of our PI-SRMs and compared them with the common NHPP-based SRMs with the same baseline intensity functions. Finally, we have verified that our SRMs performed well in all data sets and had the excellent potential ability on prediction. By carefully checking the regression coefficients, we have also confirmed that failure identification work was the most important testing metric that could contribute to software debugging, and could improve the goodness-of-fit and predictive performances.

In the future, we will propose other PI-SRMs with different baseline intensity functions. In software engineering, the measurement of software metrics and development efforts has been considered as the most fundamental technique to quantify the software product quality. However, comparing with the traditional software reliability modeling with only software fault count data, the metrics-based software reliability quantification has not been fully studied yet. In the NHPP-based modeling framework, it is important to find out the best parametric model, such as the intensity function. A similar attempt should be made in finding the best baseline intensity function, which depends on the kind of software metrics used in the analysis.

On the other hand, to analyze the classical software fault count data, we applied only the three time-dependent software metrics mentioned in [39]; fault identification effort, execution time, and computer time fault identification, while some metrics that are more easily observed as time-dependent or non-time-dependent during the testing of software engineering (e.g., the total number of operators, number of program volume, number of lines of comments, number of lines of code, number of lines of executable source code) are ignored. Therefore, we will continue to investigate our PI-SRMs in the near future by using the above mentioned metrics data as well as software fault count data.

Author Contributions: Conceptualization, Li, S., Dohi, T. and Okamura, H.; methodology, Li, S., Dohi, T. and Okamura, H.; validation, Li. S., Dohi, T. and Okamura, H. writing—original draft preparation, Li, S.; writing—review and editing, Dohi, T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lyu (ed.), M. Handbook of Software Reliability Engineering; McGraw Hill: New York, 1996.
2. Musa, J.D.; Iannino, A.; Okumoto, K. Software Reliability Measurement, Prediction, Application; McGraw-Hill: New York, 1987.
3. Pham, H. Software Reliability; Springer-Verlag: London, 2000.
4. Goel, A.L.; Okumoto, K. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability **1979**, R-28, 206–211.
5. Yamada, S.; Ohba, M.; Osaki, S. S-shaped reliability growth modeling for software error detection. IEEE Transactions on Reliability **1983**, R-32, 475–478.
6. Zhao, M.; Xie, M. On maximum likelihood estimation for a general non-homogeneous Poisson process. Scandinavian Journal of Statistics **1996**, 23, 597–607.
7. Abdel-Ghaly, A.A.; Chan, P.Y.; Littlewood, B. Evaluation of competing software reliability predictions. IEEE Transactions on Software Engineering **1986**, SE-12, 950–967.
8. Ohba, M. Inflection S-shaped software reliability growth model. In Stochastic Models in Reliability Theory; Springer, 1984; pp. 144–162.
9. Gokhale, S.S.; Trivedi, K.S. Log-logistic software reliability growth model. Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (HASE 1998), 1998, pp. 34–41.
10. Okamura, H.; Dohi, T.; Osaki, S. Software reliability growth models with normal failure time distributions. Reliability Engineering & System Safety **2013**, 116, 135–141.
11. Achcar, J.A.; Dey, D.K.; Niverthi, M. A Bayesian approach using nonhomogeneous Poisson processes for software reliability models. In Frontiers in Reliability; World Scientific, 1998; pp. 1–18.
12. Ohishi, K.; Okamura, H.; Dohi, T. Gompertz software reliability model: estimation algorithm and empirical validation. Journal of Systems and Software **2009**, 82, 535–543.
13. Rinsaka, K.; Shibata, K.; Dohi, T. Proportional Intensity-Based Software Reliability Modeling with Time-Dependent Metrics. 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006, Vol. 1, pp. 369–376.
14. Shibata, K.; Rinsaka, K.; Dohi, T. PISRAT: Proportional Intensity-Based Software Reliability Assessment Tool. 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), 2007, pp. 43–52.
15. Okamura, H.; Dohi, T. SRATS: software reliability assessment tool on spreadsheet (Experience report). 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE 2013), 2013, pp. 100–107.
16. McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering **1976**, SE-2, 308–320.
17. Halstead, M.H. Elements of Software Science; Elsevier: New York, 1977.
18. Putnam, L.H. A general empirical solution to the macro software sizing and estimating problem. IEEE Transactions on Software Engineering **1978**, SE-4, 345–367.
19. Takahashi, M.; Kamayachi, Y. An empirical study of a model for program error prediction. Proc. 8th Int'l Conf. on Software Eng. ACM/IEEE CS Press, 1985, pp. 330–336.
20. Pillai, K.; Nair, V.S.S. A model for software development effort and cost estimation. IEEE Transactions on Software Engineering **1997**, 23, 485–497.
21. Khoshgoftaar, T.M.; Munson, J.C. Predicting software development errors using software complexity metrics. IEEE Journal of Selected Areas in Communications **1990**, 8, 253–261.
22. Khoshgoftaar, T.M.; Bhattacharyya, B.B.; Richardson, G.D. Predicting software errors, during development, using nonlinear regression models: a comparative study. IEEE Transactions on Reliability **1992**, 41, 390–395.

23. Khoshgoftaar, T.M.; Munson, J.C.; Bhattacharyya, B.B.; Richardson, G.D. Predictive modeling techniques of software quality from software measures. IEEE Transactions on Software Engineering **1992**, 18, 979–987.
24. Khoshgoftaar, T.M.; Pandya, A.; Lanning, D. Application of neural networks for predicting program fault. Annals of Software Engineering. **1995**, 1, 141–154.
25. Schnieidewind, N.F. Software metrics model for integrating quality control and prediction. Proc. 8th Int'l Sympo. on Software Reliab. Eng., 1997, pp. 402–415.
26. Schneidewind, N.F. Measuring and evaluating maintenance process using reliability, risk, and test metrics. IEEE Transactions on Software Engineering **1999**, 25, 768–781.
27. Li, P.L.; Shaw, M.; Herbsleb, J.; Ray, B.; Santhanam, P. Empirical evaluation of defect projection models for widely-deployed production software systems. Proc. 12th ACM SIGSOFT Sympo. on Foundations of Software Eng. ACM, 2004, pp. 263–272.
28. Khoshgoftaar, T.M.; Gao, K.; Szabo, R. Comparing software fault predictions of pure and zero-inflated Poisson regression models. International Journal of Systems Science **2005**, 36, 705–715.
29. Amasaki, S.; Yoshitomi, T.; Mizuno, O.; Takagi, Y.; Kikuno, T. A new challenge for applying time series metrics data to software quality estimation. Software Quality Journal **2005**, 13, 177–193.
30. Ascher, H. Proportional hazards modelling of software failure data. Software Reliability; State of the Art Report (A. Bendell and P. Mellor, eds.) **1986**, pp. 229–263.
31. Ascher, H. The use of regression techniques for matching reliability models to the real world. Software System Design Methods, NATO ASI Series (J. K. Skwirzynski, ed.) **1986**, F22, 366–378.
32. Bendell, A. The use of exploratory data analysis techniques for software reliability assessment and prediction. Software System Design Methods, NATO ASI Series (J. K. Skwirzynski, ed.) **1986**, F22, 337–351.
33. Evanco, W.M.; Lacovara, R. A model-based framework for the integration of software metrics. Journal of Systems and Software **1995**, 26, 75–84.
34. Evanco, W.M. Using a proportional hazards model to analyze software reliability. Proc. 9th Int'l Conf. Software Technology & Engineering Practice. IEEE CS Press, 1999, pp. 134–141.
35. Nishio, Y.; Dohi, T. Determination of the optimal software release time based on proportional hazards software reliability growth models. Journal of Quality in Maintenance Engineering **2003**, 9, 48–65.
36. Cox, D.R. Regression models and life-tables. Journal of the Royal Statistical Society **1972**, B-34, 187–220.
37. Murphy, S.; Sen, P. Time-dependent coefficients in a Cox type regression model. Stochastic Processes and Their Applications **1991**, 39, 153–180.
38. L. Tian, D.Z.; Wei, L.J. On the Cox model with time-varying regression coefficient. Journal of the American Statistical Association **2005**, 100, 172–183.
39. Musa, J.D. Software Reliability Data, Technical Report, Data and Analysis Center for Software; Rome Air Development Center: New York, 1979.
40. Shibata, K.; Rinsaka, K.; Dohi, T. Metrics-Based Software Reliability Models Using Non-homogeneous Poisson Processes. 2006 17ths International Symposium on Software Reliability Engineering, 2006, pp. 52–61.
41. Okamura, H.; Etani, Y.; Dohi, T. A Multi-factor Software Reliability Model Based on Logistic Regression. 2010 IEEE 21st International Symposium on Software Reliability Engineering, 2010, pp. 31–40.
42. Kuwa, D.; Dohi, T. Generalized Logit Regression-Based Software Reliability Modeling with Metrics Data. 2013 IEEE 37th Annual Computer Software and Applications Conference, 2013, pp. 246–255.
43. Kuwa, D.; Dohi, T.; Okamura, H. Generalized Cox Proportional Hazards Regression-Based Software Reliability Modeling with Metrics Data. 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing, 2013, pp. 328–337.
44. Nagaraju, V.; Jayasinghe, C.; Fiondella, L. Optimal test activity allocation for covariate software reliability and security models. Journal of Systems and Software **2020**, 168, 110643.
45. Lawless, J.F.L. Regression methods for Poisson process data. Journal of the American Statistical Association **1987**, 82, 808–815.
46. Goel, A.L. Software reliability models: assumptions, limitations, and applicability. IEEE Transactions on Software Engineering **1985**, SE-11, 1411–1423.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Table 5. Predictive performance based on PMSE at 50% observation point.

GDS1		
	Best model	PMSE
Case I (cumulative)	tlogist-III	6.409
Case I (non-cumulative)	tlogist-II	4.014
Case II (cumulative)	**lxvmax-II**	**2.160**
Case II (non-cumulative)	txvmax-IV	4.931
Case III (cumulative): Linear regression	exp-IV	4.146
Case III (cumulative): Exponential regression	txvmin-V	19.213
Case III (non-cumulative): Linear regression	txvmax-II	3.916
SRATS	tnorm	3.408
GDS2		
	Best model	PMSE
Case I (cumulative)	tlogist-II	0.816
Case I (non-cumulative)	tnorm-III	0.799
Case II (cumulative)	gamma-II	0.742
Case II (non-cumulative)	**txvmax-II**	**0.407**
Case III (cumulative): Linear regression	tlogist-IV	0.616
Case III (cumulative): Exponential regression	tnorm-III	1.644
Case III (non-cumulative): Linear regression	tlogist-IV	0.780
SRATS	tlogist	1.769
GDS3		
	Best model	PMSE
Case I (cumulative)	tlogist-II	2.676
Case I (non-cumulative)	txvmax-III	0.481
Case II (cumulative)	**exp-VII**	**0.467**
Case II (non-cumulative)	pareto-VI	1.506
Case III (cumulative): Linear regression	llogist-II	0.748
Case III (cumulative): Exponential regression	lxvmax-VI	1.842
Case III (non-cumulative): Linear regression	lxvmax-VII	1.769
SRATS	exp	1.836
GDS4		
	Best model	PMSE
Case I (cumulative)	tlogist-III	2.088
Case I (non-cumulative)	pareto-II	1.506
Case II (cumulative)	exp-I	0.495
Case II (non-cumulative)	**tnorm-VI**	**0.425**
Case III (cumulative): Linear regression	txvmax-VI	1.139
Case III (cumulative): Exponential regression	exp-II	0.688
Case III (non-cumulative): Linear regression	lxvmin-I	0.703
SRATS	tlogist	1.754

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Table 6. Predictive performance based on PMSE at 80% observation point.

GDS1		
	Best model	PMSE
Case I (cumulative)	tnorm-II	2.482
Case I (non-cumulative)	txvmax-III	1.768
Case II (cumulative)	txvmax-VII	2.142
Case II (non-cumulative)	txvmax-V	2.903
Case III (cumulative): Linear regression	**tnorm-II**	**1.033**
Case III (cumulative): Exponential regression	tlogist-VII	3.159
Case III (non-cumulative): Linear regression	txvmax-VII	3.916
SRATS	txvmin	1.218
GDS2		
	Best model	PMSE
Case I (cumulative)	pareto-IV	0.488
Case I (non-cumulative)	**gamma-V**	**0.277**
Case II (cumulative)	lnorm-VII	0.399
Case II (non-cumulative)	pareto-I	0.466
Case III (cumulative): Linear regression	exp-IV	0.455
Case III (cumulative): Exponential regression	llogist-VI	0.499
Case III (non-cumulative): Linear regression	llogist-IV	0.508
SRATS	lnorm	0.531
GDS3		
	Best model	PMSE
Case I (cumulative)	tnorm-II	0.326
Case I (non-cumulative)	**txvmax-II**	**0.150**
Case II (cumulative)	txvmax-IV	0.330
Case II (non-cumulative)	lxvmax-II	0.982
Case III (cumulative): Linear regression	lxvmin-I	0.340
Case III (cumulative): Exponential regression	txvmin-VI	1.484
Case III (non-cumulative): Linear regression	pareto-III	0.293
SRATS	exp	0.295
GDS4		
	Best model	PMSE
Case I (cumulative)	exp-I	0.213
Case I (non-cumulative)	lxvmin-V	0.227
Case II (cumulative)	tnorm-IV	0.220
Case II (non-cumulative)	**tnorm-II**	**0.206**
Case III (cumulative): Linear regression	tlogist-II	0.207
Case III (cumulative): Exponential regression	lxvmax-III	0.273
Case III (non-cumulative): Linear regression	tlogist-VII	0.220
SRATS	gamma	0.230

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 July 2022 doi:10.20944/preprints202207.0357.v1

Table 7. Software reliability assessment with best SRM (minimum AIC).

(i) Best proportional intensity model (cumulative metrics data)		
	Model	Reliability
GDS1	tlogist-VI	2.969E-02
GDS2	tlogist-III	**9.260E-02**
GDS3	txvmin-II	**9.998E-01**
GDS4	exp-I	5.455E-03
(ii) Best proportional intensity model (non-cumulative metrics data)		
GDS1	txvmin-II	**4.393E-01**
GDS2	llogist-II	1.984E-02
GDS3	gamma-II	2.945E-01
GDS4	exp-VI	4.324E-01
(iii) Best SRATS (no metrics data)		
GDS1	tlogist	6.977E-05
GDS2	llogist	4.152E-03
GDS3	lxvmax	7.236E-05
GDS4	txvmin	**9.559E-01**