

# A software reliability model with time-dependent fault detection and fault removal

Mengmeng Zhu<sup>1</sup> · Hoang Pham<sup>1</sup>

Received: 11 February 2016 / Accepted: 12 February 2016 / Published online: 20 April 2016  
© The Author(s) 2016. This article is published with open access at Springerlink.com

**Abstract** The common assumption for most existing software reliability growth models is that fault is independent and can be removed perfectly upon detection. However, it is often not true due to various factors including software complexity, programmer proficiency, organization hierarchy, etc. In this paper, we develop a software reliability model with considerations of fault-dependent detection, imperfect fault removal and the maximum number of faults software. The genetic algorithm (GA) method is applied to estimate the model parameters. Four goodness-of-fit criteria, such as mean-squared error, predictive-ratio risk, predictive power, and Akaike information criterion, are used to compare the proposed model and several existing software reliability models. Three datasets collected in industries are used to demonstrate the better fit of the proposed model than other existing software reliability models based on the studied criteria.

**Keywords** Non-homogeneous Poisson process (NHPP) · Software reliability growth · Fault-dependent detection · Imperfect fault removal · The maximum number of software faults

## 1 Introduction

Reliability research has been studied over the past few decades. Also, considerable research has been done in the hardware reliability field. The increasing significant impact on software has shifted our attention to software reliability,

owing to the fact that software developing cost and software failure penalty cost are becoming major expenses during the life cycle of a complex system for a company [1]. Software reliability models can provide quantitative measures of the reliability of software systems during software development processes [2]. Most software bugs only produce inconvenient experiences to customers, but some may result in a serious consequence. For instance, because of a race condition in General Energy's monitoring system, the 2003 North America blackout was triggered by a local outage. From the latest report, Toyota's electronic throttle control system (ETCS) had bugs that could cause unintended acceleration. At least 89 people were killed as a result.

Software reliability is a significant measurement to characterize software quality and determine when to stop testing and release software upon the predetermined objectives [3]. A great number of software reliability models also have been proposed in the past few decades to predict software failures and determine the release time based on a non-homogeneous Poisson process (NHPP). Some software reliability models consider perfect debugging, such as [4–7]; some assume imperfect debugging [6–10]. The fault detection rate, described by a constant [4, 6, 11] or by learning phenomenon of developers [3, 8, 12–16], is also studied in literature. However, lots of difficulties are also generated from model assumptions when applying software reliability models on real testing environment. These non-significant assumptions have limited their usefulness in the real-world application [17]. For most software reliability models in literature, software faults are assumed to be removed immediately and perfectly upon detection [9, 17, 18]. Additionally, software faults are assumed to be independent for simplicity reason. Several studies including [3, 19, 20] incorporate fault removal efficiency and imperfect debugging into the modeling consideration. Also, Kapur et al. [21] consider that the

✉ Hoang Pham  
hopham@rci.rutgers.edu

<sup>1</sup> Department of Industrial and Systems Engineering,  
Rutgers University, New Brunswick, NJ 08854, USA

delay of the fault removal efficiency depends on their criticality and urgency in the software reliability modeling in the operation phase. However, we have not seen any research incorporate fault-dependent detection and imperfect fault removal based on our knowledge.

Why do we need to address imperfect fault removal and fault-dependent detection in the software development process? Firstly, in practice, the software debugging process is very complex [14, 22]. When the software developer needs to fix the detected fault, he/she will report to the management team first, get the permission to make a change, and submit the changed code to fix a fault [17]. In most cases, we assume that the submitted fix at the first attempt is able to perfectly remove the faults. But this perfect fault removal assumption is not realistic due to the complexity of coding and different domain knowledge level for the software developer, since domain knowledge has become a significant factor in the software development process based on the newly revisited software environmental survey analysis [14].

Moreover, the proficiency of domain knowledge has a direct impact on the fault detection and removal efficiency [14]. If the submitted fix cannot completely remove the fault, the number of actual remaining errors in the software is higher than what we estimated. These remaining faults contained in the software, indeed, affect the quality of the product. The company will release the software based on the scheduled date and pre-determined software reliability value; however, the actual quality of the software product is not as good as what we expect. Hence, the amount of complaints from the end-user may be above expectations, and the penalty cost to fix the faults through an operation phase will be much higher than in an in-house environment [23]. In consequence, the software organization has to release the updated version of the software product earlier than determined, if they want to lower the fixing cost of software faults existing in the current release. Therefore, it is plausible to incorporate imperfect fault removal into software reliability modeling for consideration in the long run.

Furthermore, software faults can be classified depending on the kind of failures they induce [24]. For instance, Bohrbug is defined as a design error and always causes a software failure when the operation system is functioning [25]. Bohrbugs are easy to be detected and removed in the very early stage of software developing or testing phase [26]. In contrast, a bug that is complex and obscure and may cause chaotic or even non-deterministic behaviors is called Mandelbug [27]. Often, Mandelbug is triggered by a complex condition [28], such as an interaction of hardware and software, or a different application field. Thus, it is difficult to remove or completely remove Mandelbugs in the testing phase due to the non-deterministic behaviors of Mandelbugs [26]. Most importantly, the detection of this type of fault is not independent and relies on the previously detected software errors. Hence,

including fault-dependent detection is desirable in model development. Of course, imperfect fault removal provides more realistic explanation for this type of software faults.

What is the maximum number of software faults and why do we incorporate them in this study? Due to the fact that software fault removal is not perfect in reality and new faults will be introduced in debugging, there is always a portion of software faults left in the software product after every debugging effort. These non-removed software faults and newly generated faults, which are caused by the interaction of new faults and existing faults, cumulate in the current version and will be carried into the next phase. Hence, the maximum number of software faults is defined in this paper, also an unknown parameter, which can be interpreted as the maximum number of faults that the software product can carry while under the designed function. If the number of faults that the software contains is larger than the maximum number, the software product will stop the designed function.

In this paper, we propose a model with considerations of fault-dependent detection, imperfect fault removal, the maximum number of software faults and logistic failure growth to predict software failures and estimate software reliability during the software development process.

Section 2 describes the software reliability modeling development and the interpretation of practical application. Section 3 states four goodness-of-fit criteria. Section 4 compares the proposed model with existing software reliability models based on the goodness-of-fit criteria described in Sect. 3, using three datasets collected from real software applications. Section 5 draws the conclusion of the proposed model and points out the future research with the application of the proposed model.

**Notation**  $N(t)$  The total number of software failures by time  $t$  based on NHPP.  
 $m(t)$  The expected number of software failures by time  $t$ , i.e.,  $m(t) = E[N(t)]$ .  
 $a(t)$  Fault content function.  
 $L$  The maximum number of faults software is able to contain.  
 $b(t)$  Software fault detection rate per fault per unit of time.  
 $c(t)$  Non-removed error rate per unit of time.  
 $\lambda(t)$  Failure intensity function.  
 $R(x|t)$  Software reliability function by time  $x$  given a mission time  $t$ .

## 2 Software reliability modeling

It is commonly assumed that the software failure intensity is proportional to the remaining faults contained in the software in most existing NHPP models. Moreover, software faults are independent and can be removed perfectly upon detection.

**Table 1** Summary of software reliability models

Model name	Model type	MVF (m(t))
Goel–Okumoto (G–O) model [4]	Concave	$m(t) = a(1 - e^{bt})$
Delayed S-shaped model [6]	S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$
Inflection S-shaped model [7]	S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$
Yamada imperfect debugging model [29]	Concave	$m(t) = a [1 - e^{-bt}] [1 - \frac{\alpha}{b}] + \alpha at$
PNZ model [8]	S-shaped and concave	$m(t) = \frac{a[(1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha t]}{1 + \beta e^{-bt}}$
Pham-Zhang model [30]	S-shaped and concave	$m(t) = \frac{1}{1 + \beta e^{-bt}} [(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$
Dependent-parameter model [31]	S-shaped and concave	$m(t) = \alpha(1 + \gamma t)(\gamma t + e^{-\gamma t} - 1)$
Dependent-parameter model with $m(t_0) \neq 0, t_0 \neq 0$ [32]	S-shaped and concave	$m(t) = m_0(\frac{\gamma t + 1}{\gamma t_0 + 1})e^{-\gamma(t - t_0)} + \alpha(\gamma t + 1)[\gamma t - 1 + (1 - \gamma t_0)e^{-\gamma(t - t_0)}]$
Loglog fault-detection rate model [18]	Concave	$m(t) = N(1 - e^{-(a^b - 1)})$
Proposed model	S-shaped and concave	$m(t) = \frac{\beta + e^{bt}}{\frac{b}{L(b-c)} [e^{bt} - e^{ct}] + \frac{1 + \beta}{m_0} e^{ct}}$

Therefore, a general NHPP software mean value function (MVF) by considering time-dependent failure detection rate is given as

$$m(t) = N \left[ 1 - e^{-\int_0^t b(x) dx} \right]. \tag{1}$$

However, nowadays, the above software mean value function is not adequate to describe more complex software product. In reality, software faults cannot be completely removed upon detection due to the programmer’s level of proficiency and the kind of faults. Some software faults may not appear in the testing phase, but could manifest in the operation field. The detection for software faults is not independent, but depends on the previously detected errors. Debugging is an error-removal process, as well as an error-induction process. In other words, we can treat it as a fault-growing process, since all the leftover (unremovable) faults cumulate in the software. Hence, a maximum number of software faults are introduced.

An NHPP software reliability model, which not only considers fault-dependent detection and imperfect fault removal process, but also takes into account the maximum number of faults in the software, is proposed in this paper. The assumptions for this proposed NHPP model are given as follows:

1. The software failure process is a non-homogeneous Poisson process (NHPP).
2. This is a fault-dependent detection process.
3. Fault detection is a learning curve phenomenon process.
4. Fault is not removed perfectly upon detection.

5. The debugging process may introduce new errors into the software. This is an imperfect debugging process, but the maximum faults contained in the software is  $L$ .
6. The software failure intensity  $\lambda(t)$  is explained as the percentage of the removed errors in the software product.
7. The non-removed software error rate is assumed to be a constant.

An NHPP software reliability model with fault-dependent detection, imperfect fault removal and the maximum number of faults can be formulated as follows:

$$\frac{dm(t)}{dt} = b(t)m(t) \left[ 1 - \frac{m(t)}{L} \right] - c(t)m(t). \tag{2}$$

The marginal condition for the above equation is given as

$$m(t_0) = m_0, \quad m_0 > 0. \tag{3}$$

Usually, the software tester performs a pre-analysis test to eliminate the most trivial errors before officially starting the testing phase. Thus, most of the existing models consider  $m_0 = 0$ . In this paper, we assume  $m_0 > 0$  by taking into consideration those trivial errors.

The general solution for (2) can be easily obtained:

$$m(t) = \frac{e^{\int_{t_0}^t (b(\tau) - c(\tau)) d\tau}}{\frac{1}{L} \int_{t_0}^t e^{\int_{t_0}^s (b(s) - c(s)) ds} b(\tau) d\tau + \frac{1}{m_0}}, \tag{4}$$

where  $m(t)$  represents the expected number of software failures detected by time  $t$ ,  $L$  denotes the maximum number of software faults,  $b(t)$  is the fault detection rate per individual

**Table 2** Phase I system test data [34]

Week index	Exposure time (cumulative system test hours)	Fault	Cumulative fault
1	356	1	1
2	712	0	1
3	1068	1	2
4	1424	1	3
5	1780	2	5
6	2136	0	5
7	2492	0	5
8	2848	3	8
9	3204	1	9
10	3560	2	11
11	3916	2	13
12	4272	2	15
13	4628	4	19
14	4984	0	19
15	5340	3	22
16	5696	0	22
17	6052	1	23
18	6408	1	24
19	6764	0	24
20	7120	0	24
21	7476	2	26

**Table 3** Phase II system test data [34]

Week index	Exposure time (cumulative system test hours)	Fault	Cumulative fault
1	416	3	3
2	832	1	4
3	1248	0	4
4	1664	3	7
5	2080	2	9
6	2496	0	9
7	2912	1	10
8	3328	3	13
9	3744	4	17
10	4160	2	19
11	4576	4	23
12	4992	2	25
13	5408	5	30
14	5824	2	32
15	6240	4	36
16	6656	1	37
17	7072	2	39
18	7488	0	39
19	7904	0	39
20	8320	3	42
21	8736	1	43

fault per unit of time and  $c(t)$  represents the non-removed error rate per unit of time.

$(1 - \frac{m(t)}{L})$  indicates the proportion of available resources that can be used in the future, which can also be interpreted as the proportion of software faults detected in every debugging effort.  $b(t)m(t) \left[1 - \frac{m(t)}{L}\right]$  is the percentage of detected dependent errors by time  $t$ .  $c(t)m(t)$  represents the non-removed errors by time  $t$ . Hence,  $b(t)m(t) \left[1 - \frac{m(t)}{L}\right] - cm(t)$  represents the proportion of the removed errors in the software by time  $t$ .  $\lambda(t) = \frac{dm(t)}{dt}$  is the failure intensity function for the whole software system by time  $t$ .

Assume that fault detection is a learning process which can be addressed in Eq. (5) and non-removed rate  $c(t)$  is a constant,

$$b(t) = \frac{b}{1 + \beta e^{-bt}}, \quad b > 0, \quad \beta > 0, \quad (5)$$

$$c(t) = c, \quad c > 0. \quad (6)$$

Substitute (5) and (6) into (4), we obtain

$$m(t) = \frac{\beta + e^{bt}}{\frac{b}{L(b-c)} [e^{bt} - e^{ct}] + \frac{1+\beta}{m_0} e^{ct}}. \quad (7)$$

The software reliability function within  $(t, t + x)$  based on the proposed NHPP is given by

$$R(x|t) = e^{-[m(t+x) - m(t)]}. \quad (8)$$

Table 1 summarizes the features and mean value function of the proposed model and existing models.

### 3 Parameter estimation and goodness-of-fit criteria

We apply the genetic algorithm (GA) to obtain the parameter estimates of the proposed model and other models as mentioned in Table 1. To compare the goodness of fit for all models, we apply four criteria in this paper described as follows. The mean-squared error (MSE) refers to the mean value of the deviation between the prediction value and the observation value as follows:

$$\text{MSE} = \frac{\sum_{i=1}^n (\hat{m}(t_i) - y_i)^2}{n - N},$$

where  $\hat{m}(t_i)$  represents the estimated expected number of faults detected by time  $t$ ;  $y_i$  represents the observation value;

**Table 4** Parameter estimation and comparison (Phase I system test data)

Model name	MSE	PRR	PP	AIC	Parameter estimate
Goel–Okumoto (G–O)	5.944	1.818	8.165	66.211	$\hat{a} = 62.0395$ $\hat{b} = 0.0243$
Delayed S-shaped	1.609	14.546	0.981	64.230	$\hat{a} = 44.221$ $\hat{b} = 0.1007$
Inflection S-shaped	0.709	1.714	0.512	63.938	$\hat{a} = 27.247$ $\hat{b} = 0.269$ $\hat{\beta} = 17.255$
Yamada imperfect debugging	2.602	0.840	0.757	66.710	$\hat{a} = 1.8643$ $\hat{b} = 0.25$ $\hat{\alpha} = 0.8418$
PNZ Model	2.479	2.954	0.690	68.611	$\hat{a} = 1.5556$ $\hat{b} = 0.3239$ $\hat{\alpha} = 0.9689$ $\hat{\beta} = 0.9999$
Pham-Zhang model	3.429	1.982	1.187	70.617	$\hat{a} = 13.394$ $\hat{b} = 0.2671$ $\hat{\alpha} = 0.5113$ $\hat{\beta} = 9.0131$ $\hat{c} = 12.0336$
Dependent-parameter model	15.741	287.191	3.768	77.541	$\hat{\alpha} = 0.0872$ $\hat{\gamma} = 0.9523$
Dependent-parameter model with $m_0 \neq 0, t_0 \neq 0$	13.477	2.136	1.189	77.621	$\hat{\alpha} = 6206$ $\hat{\gamma} = 0.0048$ $t_0 = 1$ $m_0 = 1$
Loglog fault-detection rate model	71.241	11.736	15.475	93.592	$\hat{N} = 15.403$ $\hat{a} = 1.181$ $\hat{b} = 0.567$
Proposed model	0.630	0.408	0.526	65.777	$\hat{m}_0 = 1$ $\hat{L} = 49.7429$ $\hat{\beta} = 0.2925$ $\hat{b} = 0.6151$ $\hat{c} = 0.292$

and  $n$  and  $N$  are the number of observations and the number of parameters, respectively.

The predictive-ratio risk (PRR) represents the distance of the model estimates from the actual data against the model estimates and is defined as [33]:

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

It is noticeable that the PRR value will assign a larger penalty to a model which has underestimated the cumulative number of failures.

The predictive power (PP) measures the distance of the model estimates from the actual data against the actual data, which is defined as [34]:

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

To compare the model’s ability in terms of maximizing the likelihood function while considering the degrees of freedom, Akaike information criterion (AIC) is applied.

$$AIC = -2 \log |\text{MLF}| + 2 * N,$$

where  $N$  represents the number of parameters in the model and MLF is the maximum value of the model’s likelihood function.

For all four goodness-of-fit criteria described above, the smaller the value, the better is the goodness of fit for the software reliability model.

## 4 Model evaluation and comparison

### 4.1 Software failure data description

Telecommunication system data reported by Zhang in 2002 [34] are applied to validate the proposed model. System test

**Table 5** Parameter estimation and comparison (Phase II system test data)

Model name	MSE	PRR	PP	AIC	Parameter estimate
Goel–Okumoto (G–O)	6.607	0.687	1.099	74.752	$\hat{a} = 98295$ $\hat{b} = 5.2E - 8$
Delayed S-shaped	3.273	44.267	1.429	77.502	$\hat{a} = 62.3$ $\hat{b} = 2.85E - 4$
Inflection S-shaped	1.871	5.938	0.895	73.359	$\hat{a} = 46.6$ $\hat{b} = 5.78E - 4$ $\hat{\beta} = 12.2$
Yamada imperfect debugging	4.982	4.296	0.809	78.054	$\hat{a} = 1.5$ $\hat{b} = 1.1E - 3$ $\hat{\alpha} = 3.8E - 3$
PNZ Model	1.994	6.834	0.957	75.501	$\hat{a} = 45.99$ $\hat{b} = 6.0E - 4$ $\hat{\alpha} = 0$ $\hat{\beta} = 13.24$
Pham-Zhang model	2.119	6.762	0.952	77.502	$\hat{a} = 0.06$ $\hat{b} = 6.0E - 4$ $\hat{\alpha} = 1.0E - 4$ $\hat{\beta} = 13.2$ $\hat{c} = 45.9$
Dependent-parameter model	43.689	601.336	4.530	101.386	$\hat{\alpha} = 3.0E - 6$ $\hat{\gamma} = 0.49$
Dependent-parameter model with $m_0 \neq 0, t_0 \neq 0$	35.398	2.250	1.167	87.667	$\hat{\alpha} = 890996$ $\hat{\gamma} = 1.2E - 6$ $t_0 = 832$ $m_0 = 4$
Loglog fault-detection rate model	219.687	13.655	4.383	114.807	$\hat{N} = 231.92$ $\hat{a} = 1.019$ $\hat{b} = 0.489$
Proposed model	1.058	0.163	0.144	68.316	$\hat{m}_0 = 3$ $\hat{L} = 59.997$ $\hat{\beta} = 0.843$ $\hat{b} = 0.409$ $\hat{c} = 0.108$

data consist of two phases of test data. In each phase, the system records the cumulative number of faults by each week. 356 system test hours were observed in each week for Phase I data, as shown in Table 2; 416 system test hours were observed in each week for Phase II data, as shown in Table 3. Parameter estimate was carried out by the GA method. To provide a better comparison of our proposed model with the other existing models, we analyzed Phase I as well as Phase II system test data in this section.

## 4.2 Model comparison

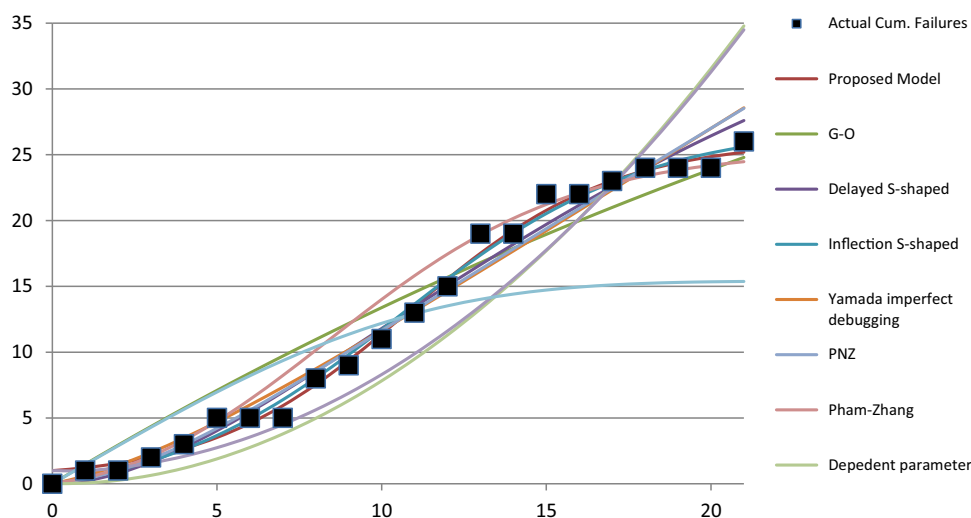
In the proposed model, when  $t = 0$ , the initial number of faults in the software satisfies  $0 < m_0 \leq y_1$ , where  $y_1$  is the number of observed failures at time  $t = 1$ ; at the same time,  $m_0$  must be an integer. The interpretation of this constraint is that the software tester often completes pre-analysis to eliminate trivial errors existing in the software before officially starting testing. The cause of these trivial errors could be

human mistakes or other simple settings. Since we consider the maximum number of faults that the software is able to contain in the modeling, these eliminated trivial errors will be counted into the total number of faults.

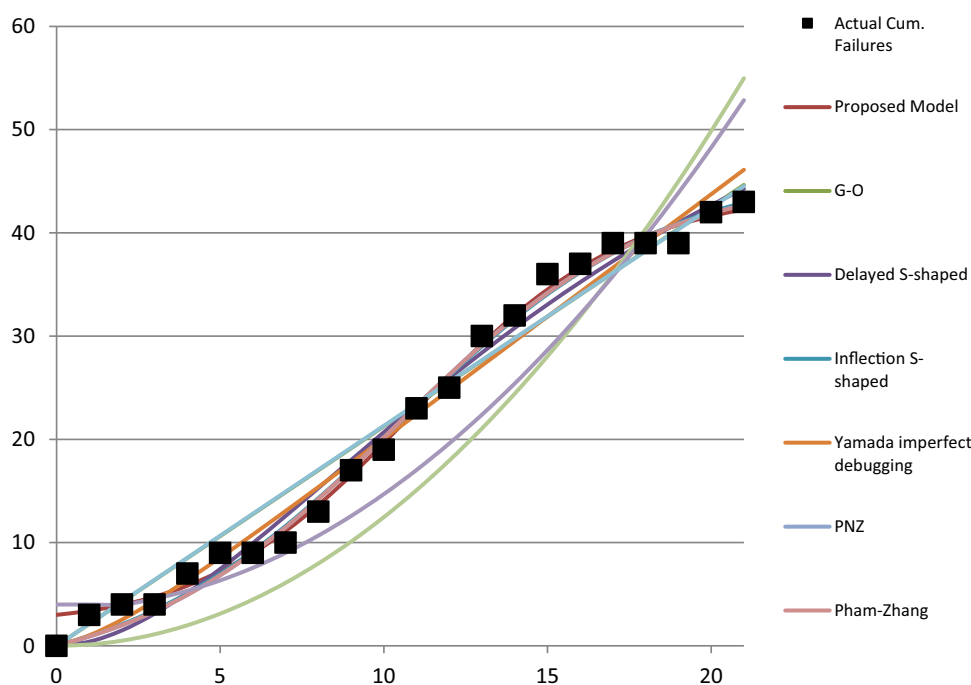
Tables 4 and 5 summarize the results of the estimated parameters and corresponding criteria value (MSE, PRR, PP, AIC) for the proposed model and other existing models. Both two-phase system test data present as an S-shaped curve; therefore, existing models such as Goel–Okumoto model is not able to perfectly capture the characteristic of the two system test datasets.

For Phase I system test data, the estimated parameters are  $\hat{m}_0 = 1$ ,  $\hat{L} = 49.7429$ ,  $\hat{\beta} = 0.2925$ ,  $\hat{b} = 0.6151$ ,  $\hat{c} = 0.292$ . As seen in Table 4, MSE and PRR values for the proposed model are 0.630 and 0.408, which are the smallest among all ten models listed here. Inflection S-shaped model has the smallest PP value. However, the PP value for the proposed model is 0.526, which is only slightly higher than 0.512. Moreover, the PRR value for the inflection S-shaped model is much higher than that of the proposed model. The

**Fig. 1** Comparison of actual cumulative failures and cumulative failures predicted by software reliability models (Phase I system test data)



**Fig. 2** Comparison of actual cumulative failures and cumulative failures predicted by software reliability models (Phase II system test data)



AIC value for the proposed model is 65.777, which is just slightly higher than the smallest AIC value, 63.938. Thus, we conclude that the proposed model is the best fit for Phase I system test data compared with the other nine models in Table 1. Figure 1 shows the comparison of actual cumulative failures and cumulative failures predicted by ten software reliability models.

For Phase II system test data, the estimated parameters are  $\hat{m}_0 = 3$ ,  $\hat{L} = 59.997$ ,  $\hat{\beta} = 0.843$ ,  $\hat{b} = 0.409$ ,  $\hat{c} = 0.108$ . The proposed model presents the smallest MSE, PRR, PP and AIC value in Table 5. Thus, we conclude that the proposed model is the best fitting for Phase II test data among all other

models in Table 1. Figure 2 plots the comparison of the actual cumulative failures and cumulative failures predicted by ten software reliability models.

Moreover, the proposed model provides the maximum number of faults contained in software, for instance,  $L = 60$  for Phase II test data. Assume that the company releases software at week 21, 43 faults will be detected upon this time based on the actual observations; however, the fault may not be perfectly removed upon detection as discussed in Sect. 2. The remaining faults shown in the operation field, mostly, are Mandelbugs [27]. Given the maximum number of faults in the software, it is very helpful for the software developer

**Table 6** Comparison of G-O, Zhang–Teng–Pham model and the proposed model using tandem computer software failure data

Testing time (weeks)	CPUhours	Defects found	Predicted total defects by G-O	Predicted total defects by Zhang–Teng–Pham model	Predicted total defects by proposed model
1	519	16	–	–	–
2	968	24	–	–	–
3	1430	27	–	–	–
4	1893	33	–	–	–
5	2490	41	–	–	–
6	3058	49	–	–	–
7	3625	54	–	–	–
8	4422	58	–	–	–
9	5218	69	–	–	–
10	5823	75	98	74.7	75.5
11	6539	81	107	80.1	80.8
12	7083	86	116	85.2	85.1
13	7487	90	123	90.1	88.5
14	7846	93	129	94.6	91.2
15	8205	96	129	98.9	93.2
16	8564	98	134	102.9	94.7
17	8923	99	139	106.8	95.8
18	9282	100	138	110.4	96.6
19	9641	100	135	111.9	97.2
20	10,000	100	133	112.2	97.6
Predicted MSE			1359.222	82.66	10.120
Predicted AIC			149.60	186.468	169.667
Predicted PRR			0.756	0.041	0.007
Predicted PP			1.395	0.050	0.006

to better predict the remaining errors and decide the release time for the next version.

#### 4.3 Software failure data from a tandem computer project

Wood [35] provides software failure data including four major releases of software products at Tandem Computers. Eight NHPP models were studied in Wood [35] and it was found that the G-O models provided the best performance in terms of goodness of fit. By fitting our model into the same subset of data, from week 1 to week 9, we predict the cumulative number of faults from week 10 to week 20, and compare the results with the G-O model and Zhang–Teng–Pham model [3]. Table 6 describes the predicted number of software failures from each model. The AIC value for the proposed model is not the smallest AIC value present in Table 6; however, we still conclude that the proposed model is the best fit for this dataset, since the other three criteria (MSE, PRR and PP) indicate that the proposed model is significantly better than other models.

The GA method is applied here to estimate the parameter. Parameter estimates for the proposed model are given as  $\hat{m}_0 = 3$ ,  $\hat{L} = 181$ ,  $\hat{\beta} = 0.5001$ ,  $\hat{b} = 0.602$ ,  $\hat{c} = 0.274$ .

## 5 Conclusions

In this paper, we introduce a new NHPP software reliability model that incorporates fault-dependent detection and imperfect fault removal, along with the maximum number of faults contained in the software. In light of the proficiency of the programmer, software faults classification, and programming complexity, we consider software fault-dependent detection and imperfect fault removal process. To our knowledge, however, not many researches have been done on estimating the maximum number of faults that can be carried in a software. We estimate the maximum number of faults in the software considering fault-dependent detection and imperfect fault removal to provide software measurement metrics, such as remaining errors, failure rate and software reliability. Hence, when to release the software and how to arrange multi-release



for a software product will be addressed in future research, since we have obtained the maximum number of faults in this study.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

- Pham, H.: Software reliability and cost models: perspectives, comparison, and practice. *Eur. J. Oper. Res.* **149**(3), 475–489 (2003)
- Pham, H., Zhang, X.: NHPP software reliability and cost models with testing coverage. *Eur. J. Oper. Res.* **145**, 443–454 (2003)
- Zhang, X., Teng, X., Pham, H.: Considering fault removal efficiency in software reliability assessment. In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 33, no. 1, pp. 114–120 (2003)
- Goel, A.L., Okumoto, K.: Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. Reliab.* **28**(3), 206–211 (1979)
- Hossain, S.A., Dahiya, R.C.: Estimating the parameters of a non-homogeneous Poisson-process model for software reliability. *IEEE Trans. Reliab.* **42**(4), 604–612 (1993)
- Ohba, M., Yamada, S.: S-shaped software reliability growth models. In: *International Colloquium on Reliability and Maintainability*, 4th, Tregastel, France, pp. 430–436 (1984)
- Ohba, M.: Inflection S-shaped software reliability growth model. *Stochastic Models in Reliability Theory*, pp. 144–162. Springer (1984)
- Pham, H., Nordmann, L., Zhang, X.: A general imperfect-software-debugging model with S-shaped fault-detection rate. *IEEE Trans. Reliab.* **48**(2), 169–175 (1999)
- Xie, M., Yang, B.: A study of the effect of imperfect debugging on software development cost. *IEEE Trans. Softw. Eng.* **29**(5), 471–473 (2003)
- Pham, H., Zhang, X.: An NHPP software reliability model and its comparison. *Int. J. Reliab. Qual. Saf. Eng.* **4**(3), 269–282 (1997)
- Pham, L., Pham, H., et al.: Software reliability models with time-dependent hazard function based on Bayesian approach. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **30**(1), 25–35 (2000)
- Pham, H., Wang, H.: A quasi-renewal process for software reliability and testing costs. In: *IEEE Transaction on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 31, no. 6, pp. 623–631 (2001)
- Jones, C.: Software defect-removal efficiency. *Computer* **29**(4), 94–95 (1996)
- Zhu, M., Zhang, X., Pham, H.: A comparison analysis of environmental factors affecting software reliability. *J. Syst. Softw.* **109**, 150–160 (2015)
- Pham, H., Pham, D.H., Pham, H.: A new mathematical logistic model and its applications. *Int. J. Inf. Manag. Sci.* **25**(2), 79–99 (2014)
- Fang, C.-C., Chun-Wu, Y.: Effective confidence interval estimation of fault-detection process of software reliability growth models. *J. Syst. Sci* (2015). doi:10.1080/00207721.2015.1036474
- Ho, S.L., Xie, M., Goh, T.N.: A study of the connectionist models for software reliability prediction. *Comput. Math. Appl.* **46**(7), 1037–1045 (2003)
- Pham, H.: Loglog fault-detection rate and testing coverage software reliability models subject to random environments. *Vietnam J. Comput. Sci.* **1**(1), 39–45 (2014)
- Zhang, X., Pham, H.: Software field failure rate prediction before software deployment. *J. Syst. Softw.* **79**(3), 291–300 (2006)
- Zhang, X., Jeske, D.R., Pham, H.: Calibrating software reliability models when the test environment does not match the user environment. *Appl. Stoch. Models Bus. Ind.* **18**(1), 87–99 (2002)
- Kapur, P.K., Gupta, A., Jha, P.C.: Reliability analysis of project and product type software in operational phase incorporating the effect of fault removal efficiency. *Int. J. Reliab. Qual. Saf. Eng.* **14**(3), 219–240 (2007)
- Lyu, M.R.: *Handbook of Software Reliability Engineering*. vol. 222, IEEE computer society press (1996)
- Grottke, M., Trivedi, K.S.: Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer* **40**(2), 107–109 (2007)
- Grottke, M., Trivedi, K.S.: A classification of software faults. *J. Reliab. Eng. Assoc. Jpn.* **27**(7), 425–438 (2005)
- Shetti, N.M.: Heisenbugs and Bohrbugs: Why are they different. *Techn. Ber. Rutgers, The State University of New Jersey* (2003)
- Alonso, J., Grottke, M., Nikora, A.P., Trivedi, K.S.: An empirical investigation of fault repairs and mitigations in space mission system software. In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–8 (2013)
- Vaidyanathan, K., Trivedi, K.S.: Extended classification of software faults based on aging. In: *IEEE International Symposium on Software Reliability Engineering, ISSRE* (2001)
- Carrozza, G., Cotroneo, D., Natella, R., Pietrantuono, R., Russo, S.: Analysis and prediction of mandelbugs in an industrial software system. In: *IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 262–271 (2013)
- Pham, H.: A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. *Optimization* **63**(10), 1481–1490 (2014)
- Chang, I.H., et al.: A testing-coverage software reliability model with the uncertainty of operating environments. *Int. J. Syst. Sci. Oper. Logist.* **1**(4), 220–227 (2014)
- Yamada, S., Tokuno, K., Osaki, S.: Imperfect debugging models with fault introduction rate for software reliability assessment. *Int. J. Syst. Sci.* **23**(12), 2241–2252 (1992)
- Pham, H.: An imperfect-debugging fault-detection dependent-parameter software. *Int. J. Autom. Comput.* **4**(4), 325–328 (2007)
- Pham, H., Deng, C.: Predictive-ratio risk criterion for selecting software reliability models. In: *Proceedings of the 9th International Conference on Reliability and Quality in Design*, pp. 17–21 (2003)
- Pham, H.: *System Software Reliability*. Springer (2007)
- Wood, A.: Predicting software reliability. *Computer* **29**(11), 69–77 (1996)