The Journal of Systems & Software 187 (2022) 111249

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Analysis and assessment of software reliability modeling with preemptive priority queueing policy☆

Jhih-Sin Lin, Chin-Yu Huang *, Chih-Chiang Fang

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 29 September 2021
Received in revised form 21 December 2021
Accepted 23 January 2022
Available online 1 February 2022

Keywords:
Software reliability
Preemptive priority queue
Non-preemptive priority queue
Open-source software
Software testing
Debugging

ABSTRACT

Due to the growing complexity and size of software systems in modern society, the quality requirements for computer software have become ever more stringent. In the past, many software reliability growth models (SRGMs) have been proposed to evaluate and assess the reliability and quality of software systems. Some studies have demonstrated that both infinite server queueing (ISQ) and finite server queueing (FSQ) models can be applied to describe the fault detection process (FDP) and fault correction process (FCP) of software systems. However, it has also been noted that most ISQ and FSQ models assumed and obeyed the first come first served (FCFS) rule when removing the detected faults in FDP. In practice, the detected faults generally are classified into different levels of priority and those with higher priority should be fixed earlier. That is, high-priority faults have to be removed quickly to minimize their impact on software systems. Consequently, this assumption should be properly modified or adjusted. In this paper, we proposed a preemptive priority queueing (PPQ) model that considers both a finite number of debuggers and different priority levels. In our proposed PPQ model, faults assigned higher priority would be able to preemptively acquire resources already occupied by lower priority faults. Some numerical examples based on real failure data from different open-source and closed-source software are analyzed and discussed in detail. Experimental results show that the proposed PPQ model can provide more accurate estimation capability for software reliability, compared to traditional SRGMs.

© 2022 Elsevier Inc. All rights reserved.

1. Introduction

In recent years, more and more software services have been provided in our daily applications. Many practical applications such as payment systems, automated driving system, cloud services and the Internet of Things (IoT) generally require very high reliability and stability to accomplish crucial tasks. The growing size and complexity of these applications have led to an increased demand for software quality. Software reliability is typically defined as the probability that software will provide failure-free operations in a specific environment during a fixed interval of time (Musa et al., 1987). Since the 1970s, numerous software reliability growth models (SRGMs) have been proposed to predict and estimate the reliability growth of software systems (Musa et al., 1987; Pham, 2006; Lyu, 1996). Software faults are typically defined as a defective instruction or set of related instructions that is the cause of one or more actual or potential failure types (Musa et al., 1987). Note that software faults can also be classified into two types: Heisenbugs and Bohrbugs (Vaidyanathan and Trivedi, 2005). Most published SRGMs generally assume that the software failure process can be described by a Non-Homogeneous Poisson Process (NHPP). The NHPP-type SRGMs assume that software failures occur at random times (Musa et al., 1987; Tausworthe and Lyu, 1996; Xie, 1991). Additionally, some studies have also shown that infinite server queueing (ISQ) and finite server queueing (FSQ) models are applicable for describing the fault detection process (FDP) and fault correction process (FCP) of software systems (Yang, 1996; Huang and Lin, 2006; Huang and Huang, 2008; Huang and Kuo, 2017; Inoue and Yamada, 2003). Using SRGMs, developers would be able to measure software reliability, understand the characteristics of the software and ensure its quality.

On the other hand, to construct high quality software, developers generally need to identify potential faults and remove them throughout the software testing process (Wong et al., 2010; Li et al., 2019; Kong et al., 2018; Gao and Wong, 2019). However, there is no effective approach of queueing systems to debugging because the root cause of faults is usually undetectable and hard to remove immediately. Furthermore, developers often find bug reports tedious. Thus, delays caused by fault detection and correction processes should not be ignored (Yang, 1996; Huang and Lin, 2006). To address these issues, several researchers have applied both the ISQ and FSQ models to predict software reliability

☆ Editor: W. Eric Wong.

* Corresponding author.
E-mail address: cyhuang@cs.nthu.edu.tw (C.Y. Huang).

https://doi.org/10.1016/j.jss.2022.111249
0164-1212/© 2022 Elsevier Inc. All rights reserved.

Acronyms

SRGM	Software reliability growth model
NHPP	Non-homogeneous Poisson process
ISQ	Infinite server queueing
FSQ	Finite server queueing
FCFS	First come first served
PPQ	Preemptive priority queueing
GO	Goel–Okumot
DSS	Delayed S-shaped
ISS	Inflected S-shaped
FDP	Fault detection process
FCP	Fault correction process
TEF	Testing effort function
EFSQ	Extended finite-server-queueing
OSS	Open source software
CSS	Closed source software
LSE	Least square estimation
MSE	Maximum likelihood estimation
MAE	Mean absolute error
MSE	Mean square error
AIC	Akaike information criterion

Notations

$\boldsymbol{\lambda}$	Fault-detection rate
$\boldsymbol{\mu}$	Fault-correction rate
c	Number of debuggers
$\boldsymbol{\rho}$	Traffic intensity of queueing model
$N_d(t)$	Number of faults detected by time t
Nr(t)	Number of faults corrected by time t
No(t)	Number of faults uncorrected by time t
$N_{d,h}(t)$	Number of high priority faults detected by time t
$N_{d,l}(t)$	Number of low priority faults detected by time t
$N_{r,h}(t)$	Number of high priority faults corrected by time t
$N_{r,l}(t)$	Number of low priority faults corrected by time t
p_h	Probability that an arbitrary high priority fault arrives in (0, t] and removes in (0, t]
p_l	Probability that an arbitrary low priority fault arrives in (0, t] and removes in (0, t]
Q_n	There are n faults in queueing system
q	Probability that a fault is still being debugging at time t
$S_{h,k}$	Probability that the number of high priority faults in the queueing system is k
E[N]	The total expected number of faults
$E[N_h]$	Expected number of high priority faults
$E[N_l]$	Expected number of low priority faults
$E[R_h]$	Mean response time of high priority faults
$E[R_l]$	Mean response time of low priority faults
$E[W_h]$	Mean waiting time of high priority faults
$E[W_l]$	Mean waiting time of low priority faults

and describe the processes of software testing and debugging. However, we found that most of the models usually assumed and obeyed the FCFS rule in processing the detected faults, and each fault was processed in order. Moreover, high-priority faults have to be removed quickly to minimize their impact on software applications. Consequently, this assumption should be properly modified or adjusted. Mockus et al. (2002) also examined the failure data from two major open-source projects and showed that faults with higher priority should be corrected faster than faults with lower priority. These studies suggest that there is a significant relationship between debugging time and priority levels.

In general, priority scheduling can be classified into two types: preemptive and non-preemptive. Preemptive scheduling permits the scheduler to preempt a low-priority running process if a high-priority process enters the system. Under non-preemptive scheduling, once a process goes into operation, it will not be interrupted until completion. In practice, developers usually deal with higher-priority faults more rapidly than lower-priority faults. High-priority faults should take precedence over low-priority faults to avoid operating system crashes.

Based on the above considerations, in this paper we plan to propose a preemptive priority queueing model (PPQ) to analyze real software failure data and depict the failure behavior of software systems in a more practical way (Lin, 2017). We assumed that the PPQ model is a M/M/c/∞/PR queueing model, in which the two M represent exponentially distributed. The PR is the waiting queue following the preemptive priority policy, c is the number of debuggers (a finite number), and system capacity is infinite. In our proposed model, a detected fault can correspond to a customer arriving at a system and the service channels can be viewed as the debuggers. Some numerical examples based on real failure data from open-source software (OSS) and closed-source software (CSS) are presented and analyzed. Experimental results show that the proposed PPQ model can provide more accurate results for software reliability prediction, compared to traditional SRGMs.

The organization of this paper is as follows. In Section 2, we provide a brief review of some SRGMs and ISQ models. In Section 3, we discuss and show in detail how to derive the PPQ model. Some mathematical properties of the PPQ model will also be discussed. In Section 4, three real datasets collected from OSS and CSS are used to evaluate the performance of our proposed PPQ model and some SRGMs. We present criteria for comparing models and make some comparisons. Finally, some conclusions are provided in Section 5.

2. Software reliability modeling

Detailed software errors often can cause serious damage to systems. During the testing phase developers have to identify these faults and remove them as early as possible. In the past three decades, many SRGMs have been proposed to evaluate the reliability of software (Lyu, 1996; Xie, 1991). Widely used SRGMs include: the Goel and Okumoto model, the Duane model, the Yamada delayed S-shaped (DSS) model, and the Ohba inflected S-shaped (ISS) model (Musa et al., 1987; Pham, 2006; Lyu, 1996). The logistic and Gompertz growth curves were also widely used to estimate the fault content of software systems (Xie, 1991). Practically, these SRGMs play an important role in estimating the number of faults remaining in the system and the failure rate, determining the appropriate release time of software systems, and providing an effective testing plan for reaching the desired reliability objectives.

However, it has to be noted that most research assume that faults can be removed immediately when they are detected in the testing phase. In reality, the fault removal rate usually depends on the skill and experience of the debugger (Huang and Lin, 2006). The time required to remove faults cannot be ignored in practice (Yang, 1996). In this SRGM, it is assumed that for a failure the detection of the faults causing the failure also results in the detection of a proportion of the remaining faults, without those faults causing any failure. It is worth noting that the DSS model may also be used to describe the fault correction process (Lyu, 1996; Xie, 1991). Hsu and Huang (2014) once proposed to use combinational SRGMs for improving the prediction capability of software reliability and they also proposed an enhanced genetic algorithm with several efficient operators for the weighted assignment of combinations.

Additionally, Kanoun and Laprie (1994) observed that the use of NHPP-based SRGMs during the early stages of development and validation is much less convincing. Like NHPP models, artificial neural network (ANN) models also suffer from difficulties in early prediction. As a result, research on NHPP and ANN frameworks has focused on improving early reliability prediction (Lyu, 1996; Hsu and Huang, 2014; Kanoun and Laprie, 1994; Xie et al., 1999; Hu et al., 2006a,b). Since NHPP models are Poisson-type models, they follow the property of Poisson distribution in which the mean and the variance coincide with each other. However, although many SRGMs have been proposed and analyzed, no single reliability model can capture the required amount of software operational characteristics, since these models make simplified assumptions and abstractions. That is, there is no single SRGM that is universally applicable to all situations (Lyu, 1996).

It can be noted that some research also indicates that queueing theory can be used to describe fault detection and correction activities during software development. For example, Yang (1996) proposed an infinite server queueing (ISQ) model to provide a more realistic model for evaluating software reliability. He divided the debugging process into fault detection and removal processes and assumed that the number of debuggers was infinite so that the faults did not require waiting for service. Dohi et al. (2002) presented a general method to existing SRGMs by considering the software failure occurrence process as an ISQ model. Their experimental results showed that the ISQ model was better than the general order statistics model. Balsamo et al. (2003) considered a queueing model with finite capacity queues and blocking after service (BAS) for software architecture performance prediction. Antoniol et al. (2004) also presented a method based on queueing theory and stochastic simulation to deal with the staffing levels, process management, and service level evaluation of cooperative distributed maintenance projects. The method has been fully validated on a large COBOL/JCL financial software system.

Dohi et al. (2004) also presented an ISQ model to describe dynamic software debugging behavior and showed that it can involve representative NHPP models with both a finite and an infinite number of faults. Inoue and Yamada (2006) showed that extended delayed S-shaped SRGM based on the ISQ model can easily represent the physical behavior for the fault-detection process (FDP). Kapur et al. (2010a,b) proposed a unified modeling method for applying ISQ theory based on three levels of complexity of fault assumptions. The method can be extended to several existing and new SRGMs.

However, no company or organization has infinite human resources. Huang and Huang (2008) incorporated ISQ and FSQ into a software reliability model under perfect and imperfect debugging environments. Their experimental results show that the new SRGM has a better fit to the observed data, and predicts future real failure behavior data well. Schneidewind (2010) mentioned single queue feeding multiple fault correction servers to analytical models and multiple queues feeding multiple servers to simulation models. His three objectives were as follows: (1) identify the optimal number of fault correction servers, (2) identify which faults may require excessive processing time, and (3) assign faults to the selected correction server. Zhang et al. (2011) presented a finite server queueing model that allows a joint study of FDP and FCP two processes and also integrated a generalized modified Weibull (GMW) testing effort function (TEF) into FDP and FCP. Their experimental results showed that the proposed model had a fairly accurate prediction capability to close the actual value. Additionally, Huang and Kuo (2017) proposed an extended finite-server-queueing (EFSQ) model to analyze the fault removal process of a software application. They also discussed and showed that the ISQ model is a special case of the EFSQ model under certain conditions. Yao and Zhang (2018) also presented a finite server queueing model with various distribution function (e.g., GMW, logistic, exponential) and change point under perfect and imperfect debugging environments. The validation and accuracy of the model have been carried out on real fault datasets.

On the other hand, Gokhale and Mullen (2006) showed that structuring the debugging system as n independent M/M/1 multi-priority queues can provide a realistic approximation to the queueing behavior of four different organizations. According to bug severities, faults are classified into many categories. Each category has a different priority level and response time needs. They applied two kinds of priority queue models: (1) preemptive priority queue and (2) head-of-the-line (non-preemptive) priority queue. They also introduced a mixed model that combines the two models. Moreover, Lim and Kim (2014) applied $M^x/G/1$ queueing models with non-preemptive priority policy to the software vulnerability models, where M represents exponential distribution, x denotes random batch size, which is homogeneous priority batch arrival, G denotes general service time distribution, and a single server. This model was used to predict the number of unfixed vulnerabilities at arbitrary instances and the mean waiting time before fixing. The service rate to prevent the number or accumulated degree of vulnerabilities from exceeding the predetermined level may also be estimated.

Furthermore, Zhang (2015) incorporated detection effort (DE) and correction effort (CE) into the FDP and the FCP, respectively. Furthermore, the FDP and FCP are modeled as the arrival and departure processes of the ISQ model. The experimental results obtained are in close accord with actual software failure data. Huang et al. (2009) proposed an extended ISQ model with multiple change-points to predict and assess software reliability. This model can accurately reflect the changes in the fault correction rate during the FCP and more realistically describe actual software debugging situations.

From the above discussions, it can be found that most studies assume that the detected faults are corrected on an FCFS basis (Huang and Huang, 2008; Gokhale and Mullen, 2006). However, Mockus et al. (2002) reported that in Apache and Mozilla, bugs with higher priority are fixed faster than bugs with lower priority. In the real world, the debugging team generally fixes faults based on their priority level (Laplante and Ahmad, 2009). For example, a high-priority fault such as a memory leak has to be corrected immediately. High-priority faults usually receive attention as long as there are no immediate priority faults waiting for correction. A low-priority fault, such as a typo or enhancement can wait for a long time without having a great impact on the system. Given this, in this paper we propose the PPQ model for software reliability modeling considering a finite server and priority level. Our goal is to process important and urgent faults first.

The debugging behaviors and activities of the proposed PPQ model are shown in Fig. 1. The white squares Pr-H and Pr-L represent high-priority and low-priority faults, respectively. Since

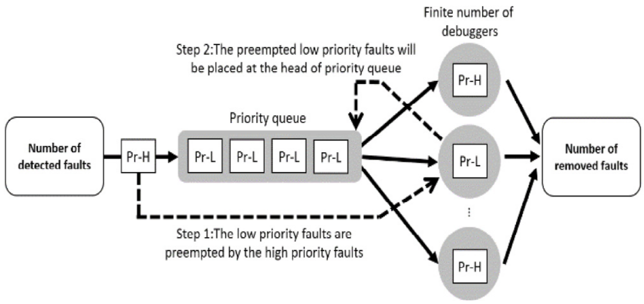

Fig. 1. Diagram of the preemptive priority queueing process during software development.

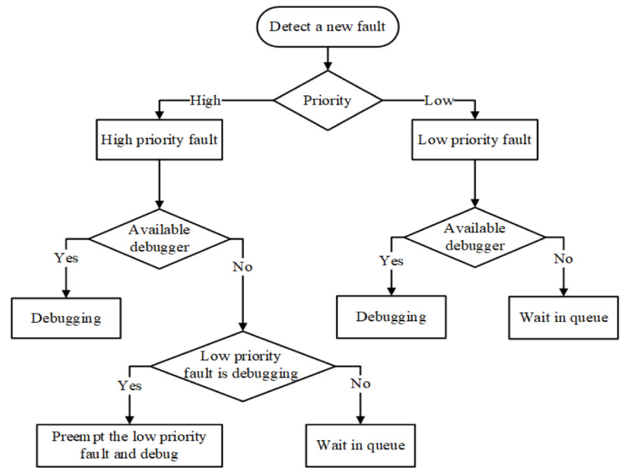

Fig. 2. Flowchart of the priority queueing model.

the number of debuggers in the system is finite, the preemption process may be triggered when all debuggers are busy. When a newly detected high-priority fault enters the system and a low-priority fault is being processed, the handling of the low-priority fault will be interrupted by the preemption process. The process consists of two steps: (1) the low-priority fault that is being processed will be preempted, and (2) the preempted fault will be returned to the head of the waiting queue until there is an available debugger. Thus, the new high-priority fault will be processed immediately. We can then ensure that high-priority faults will be corrected earlier than low-priority faults.

3. Preemptive priority queueing model

Software testing includes the process of executing test cases with the intent of finding faults and confirming specifications to meet requirements. In practice, developers need some time to diagnose and locate the root cause (or primary cause) of the failure. However, analyzing the failure logs is very time consuming. Therefore, the time between fault detection and correction should not be ignored. In this section, we propose and discuss the characteristics of the PPQ model.

3.1. The proposed PPQ model

The assumptions of the proposed PPQ model are as follows (Musa et al., 1987; Xie, 1991; Yang, 1996; Huang and Huang, 2008; Huang and Kuo, 2017; Lin, 2017):

(1) The software is subject to failures at random times caused by the manifestation of remaining faults in the system.

(2) The mean number of faults detected in the time interval (t, t + Δt) is proportional to the mean number of remaining faults in the system. Furthermore, all faults in a program are mutually independent from the failure detection point of view.

(3) The detected faults will be put into the waiting queue according to a Poisson process with rate λ. In addition, the detected faults are fixed by a finite number of debuggers c in a group. The correction time of a detected fault for each assigned debugger is exponentially distributed with rate μ.

(4) Two kinds of faults are categorized: high-priority faults and low-priority faults. High-priority faults have a preemptive ability for service over low-priority faults. If the two faults have the same priority, they will be served according to their order in the waiting queue.

(5) If all debuggers are busy with high-priority faults, a newly detected high-priority fault follows the FCFS rule and waits at the tail of the same priority faults in the waiting queue.

(6) Each time a failure occurs, the fault that causes it will eventually be removed, and no fault will be introduced. The waiting time and correction time of fault(s) are mutually independent. Fault detection activity continues while faults are removed, and fault correction does not affect the detection process.

(7) Fault correction time is non-negligible so that the number of removed faults may lag behind the total number of detected faults due to the priority and difficulty of fault correction.

Based on the above assumptions, we obtain a queueing model that describes the fault removal process, M/M/c/∞/PR (Shortle et al., 2018), assuming the finite number of debuggers is c. Fig. 2 shows the flowchart of the priority queueing model and illustrates the procedures of fault detection and correction. In general, it is acceptable that high-priority fault(s) should be corrected sooner than low-priority fault(s). Consider the situation in which there are currently n high-priority faults and m low-priority faults in the queueing system. The queue system is in state (n, m). The system goes to state (n-1, m) if a high-priority fault is corrected and removed from the system or goes to state (n, m+1) when a new low-priority fault is detected and enters the queueing system. Note that if there are less than c faults (the sum of two priorities) in the system when a fault is detected a free debugger will start correcting it immediately, no matter what its priority is. The overall system correction rate will also increase μ. However, when there are more than c high-priority faults in the system, the correction rate is at most cμ because the queueing system has a finite number of debuggers (c). In this situation, other faults must remain in the waiting queue until debuggers are available (Lin, 2017).

Moreover, the processes of fault detection and removal in a software system can be considered analogous to the arrival and service patterns of customers in a typical queueing system (Yang, 1996; Huang and Huang, 2008; Huang and Kuo, 2017). Therefore, the behavior of these processes can be described by the queueing system. Here we define random variables $N_d(t)$ and $N_r(t)$ as the number of faults detected and removed by time t, respectively (Yang, 1996; Huang and Kuo, 2017). The fault detection and removal processes can be described by these counting process, $\{N_d(t),\ t \geq 0\}$ and $\{N_r(t),\ t \geq 0\}$. The Poisson arrivals of faults can be split into multiple Poisson arrival processes. According to assumptions (3) and (4), we can divide the Poisson process of fault arrival into two independent Poisson processes, the high and low-priority fault counting processes, $\{N_{d,h}(\mathrm{t}), t \geq 0\}$ and $\{N_{d,l}(\mathrm{t}), t \geq 0\}$, respectively (Tijms, 2003). Note that $N_d(t) = N_{d,h}(t) + N_{d,l}(t)$.

Here we assume that in the time interval (0, t], there are n high-priority faults detected. Let $m_{d,h}(t)$ be the mean value function of $N_{d,h}(t)$. $m'_{d,h}(x)$ is the derivative of $m_{d,h}(x)$ with respect

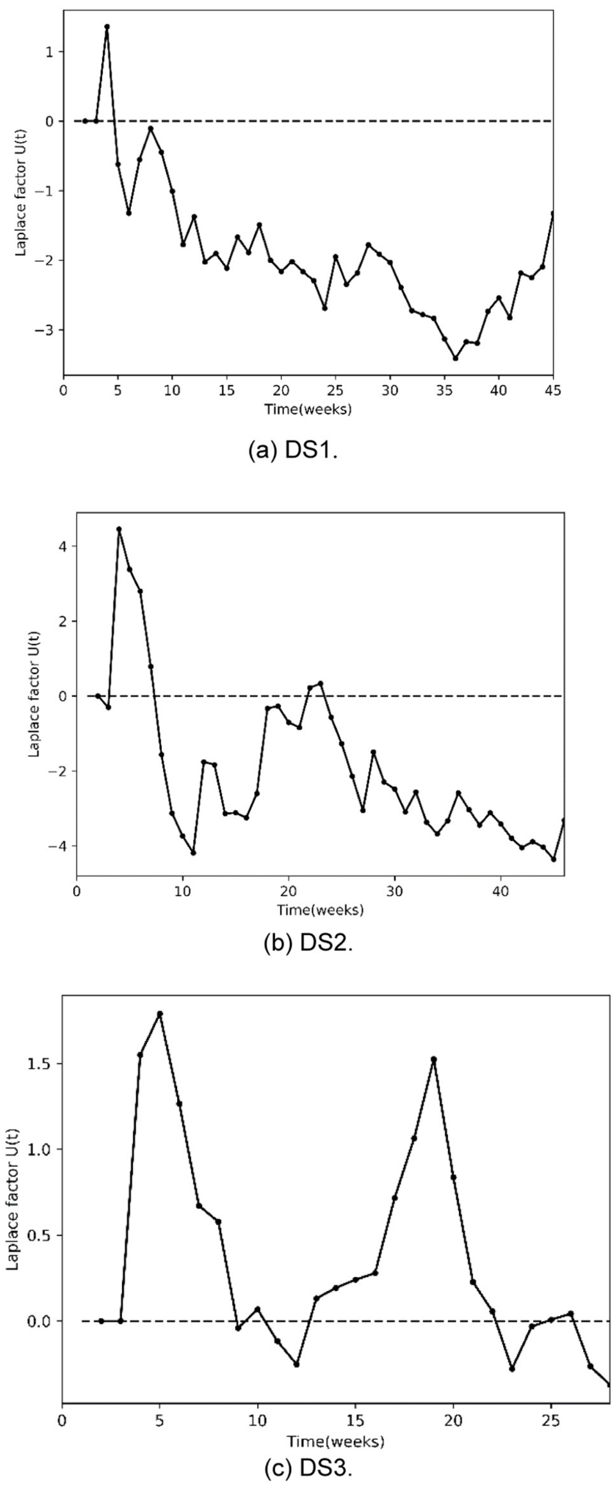

Fig. 3. Plot of the Laplace trend test for DS1, DS2, and DS3.

to x. Thus, we can obtain (Huang and Kuo, 2017):

$$P\{\text{high priority fault detected at } x\} = \frac{m'_{d,h}(x)}{m_{d,h}(t)}. \tag{1}$$

Referring to assumption (4), since a high-priority fault is able to preemptively grab the resources occupied by lower priority faults, we need only focus on the high-priority faults in this part. Let p_h be the probability that an arbitrary high-priority fault is detected in (0, t] and removed in (0, t]. By the law of total probability (Tijms, 2003), we have

$$\begin{aligned} p_h &= \int_0^t P\{removal\ time \le t - x \cap \text{arrives at } x\}dx \\ &= \int_0^t P\{removal\ time \le t - x|\text{arrives at } x\}P\{\text{arrives at} x\}dx \\ &= \int_0^t G(t-x)\frac{m'_{d,h}(x)}{m_{d,h}(t)}dx, \end{aligned} \tag{2}$$

where $G(\bullet)$ is the *cumulative distribution function* (cdf) of the debugging time. If a high-priority fault is detected, the currently processed low-priority fault will be preempted and returned to the head of the waiting queue until there is free debugger available.

Similarly, the probability that a low-priority fault is detected at time x and totally removed within (x, t] is

$$\begin{aligned} p_l &= \int_0^t P\{removal\ time \le t - x \cap \text{arrives at} x \\ &\cap \text{ not all channels busy}\}dx \\ &= \int_0^t P\{removal\ time \le t - x|\text{arrives at} x\} \\ &P\{\text{arrives at} x\}P\{\text{not all channels busy}\}dx. \end{aligned} \tag{3}$$

We let $m_{d,l}(t)$ be the mean value function of $N_{d,l}(t)$. $m'_{d,l}(x)$ is the derivative of $m_{d,l}(x)$ with respect to x. Hence, we obtain the following:

$$P\{\text{low priority fault detected at} x\} = \frac{m'_{d,l}(x)}{m_{d,l}(t)}. \tag{4}$$

The probability that not all channels are busy is as follows:

$$\begin{aligned} 1 - \sum_{n=c}^{\infty} Q_n &= 1 - Q_0 \sum_{n=c}^{\infty} \frac{(c\rho)^n}{c^{n-c}c!} \\ &= 1 - Q_0\frac{(c\rho)^c}{c!}\sum_{n=c}^{\infty}\frac{(c\rho)^{n-c}}{c^{n-c}} = 1 - \frac{Q_0(c\rho)^c}{c!(1-\rho)}, \end{aligned} \tag{5}$$

and $\rho < 1$, where Q_n is the probability that there are n number of faults in the system, ρ *is* traffic intensity, c is the number of debuggers. Referring to the probability of steady-state for the $M/M/c$ queue (Shortle et al., 2018), we obtain

$$Q_n = \begin{cases} \dfrac{(c\rho)^n}{n!}Q_0, & \text{for } 0 \le n < c, \\ \dfrac{(c\rho)^n}{c^{n-c}c!}Q_0, & \text{for } n \ge c. \end{cases} \tag{6}$$

Continuing from (6), the probability of an empty queueing system is given by

$$Q_0 = \left(\sum_{n=0}^{c-1}\frac{(c\rho)^n}{n!} + \sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!}\right)^{-1}. \tag{7}$$

Substituting (4), (5) and (7) into (3) yields

$$p_l = \int_0^t \left(1 - Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!}\right)G(t-x)\frac{m'_{d,l}(x)}{m_{d,l}(t)}dx, \tag{8}$$

From (8), two situations (variations) are discussed as follows.

Case 1: If c approaches 1, then we obtain

$$\begin{aligned} p_l &= \lim_{c\to 1}\left\{\int_0^t \left(1 - Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!}\right)G(t-x)\frac{m'_{d,l}(x)}{m_{d,l}(t)}dx\right\} \\ &= \left(1 - \lim_{c\to 1}\left\{Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!}\right\}\right)\lim_{c\to 1}\left\{\int_0^t G(t-x)\frac{m'_{d,l}(x)}{m_{d,l}(t)}dx\right\}. \end{aligned} \tag{9}$$

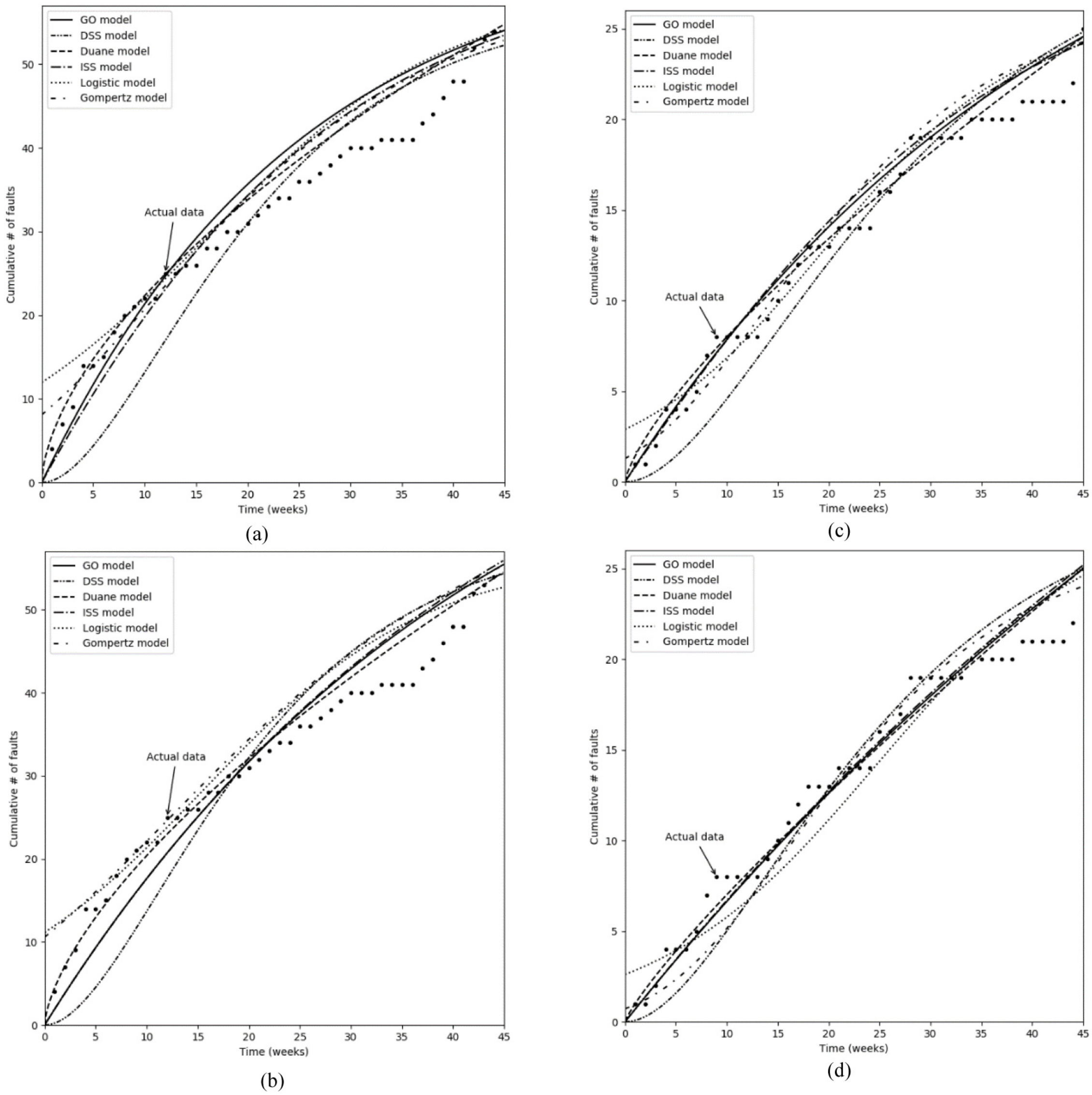

Fig. 4. Cumulative curves of actual and estimated number of detected faults (DS1): (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; (d) low priority using MLE.

We then have $p_l = 0$. In this case, the single debugger situation will cause the low-priority faults to be blocked by the high-priority faults (starvation).

Case 2: If c approaches infinity, we obtain

$$\begin{aligned} p_l &= \lim_{c\to\infty}\{\int_0^t (1 - Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!})G(t-x)\frac{m'_{d,l}(x)}{m_{d,l}(t)}dx\} \\ &= (1 - \lim_{c\to\infty}\{Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!}\})\lim_{c\to\infty}\{\int_0^t G(t-x)\frac{m'_{d,l}(x)}{m_{d,l}(t)}dx\}. \end{aligned} \tag{10}$$

In this case, the infinite debuggers eliminate the priority property of the PPQ model and p_l is equivalent to (2).

The probability that a fault is still being debugged at time t is q=1-p_h-p_l. Let $N_o(t)$ be the number of open-remaining faults at time t, and apply the binomial theorem and regard the n detection as independent trials. In this case, we have $n = i + j + k$ and

$$\begin{aligned} &P\{N_{r_h}(t) = i, N_{r_l}(t) = j, N_o(t) = k | N_d(t) = i + j + k\} \\ &= \frac{(i+j+k)!}{i!j!k!}{p_h}^i {p_l}^j q^k. \end{aligned} \tag{11}$$

Since $N_d(t)$ is Poisson distributed, we have

$$\begin{aligned} &P\{N_{r,h}(t) = i, N_{r,l}(t) = j, N_o(t) = k\} \\ &= P\{N_{r,h}(t) = i, N_{r,l}(t) = j, N_o(t) = k | N_d(t) = n\}P\{N_d = n\} \\ &= \frac{(i+j+k)!}{i!j!k!}{p_h}^i {p_l}^j q^k \frac{{m_d}^{i+j+k}}{(i+j+k)!}e^{-m_d(t)} \\ &= \frac{[m_d(t)p_h]^i}{i!}e^{-m_d(t)p_h}\frac{[m_d(t)p_l]^j}{j!}e^{-m_d(t)p_l}\frac{[m_d(t)q]^k}{k!}e^{-m_d(t)q}. \end{aligned} \tag{12}$$

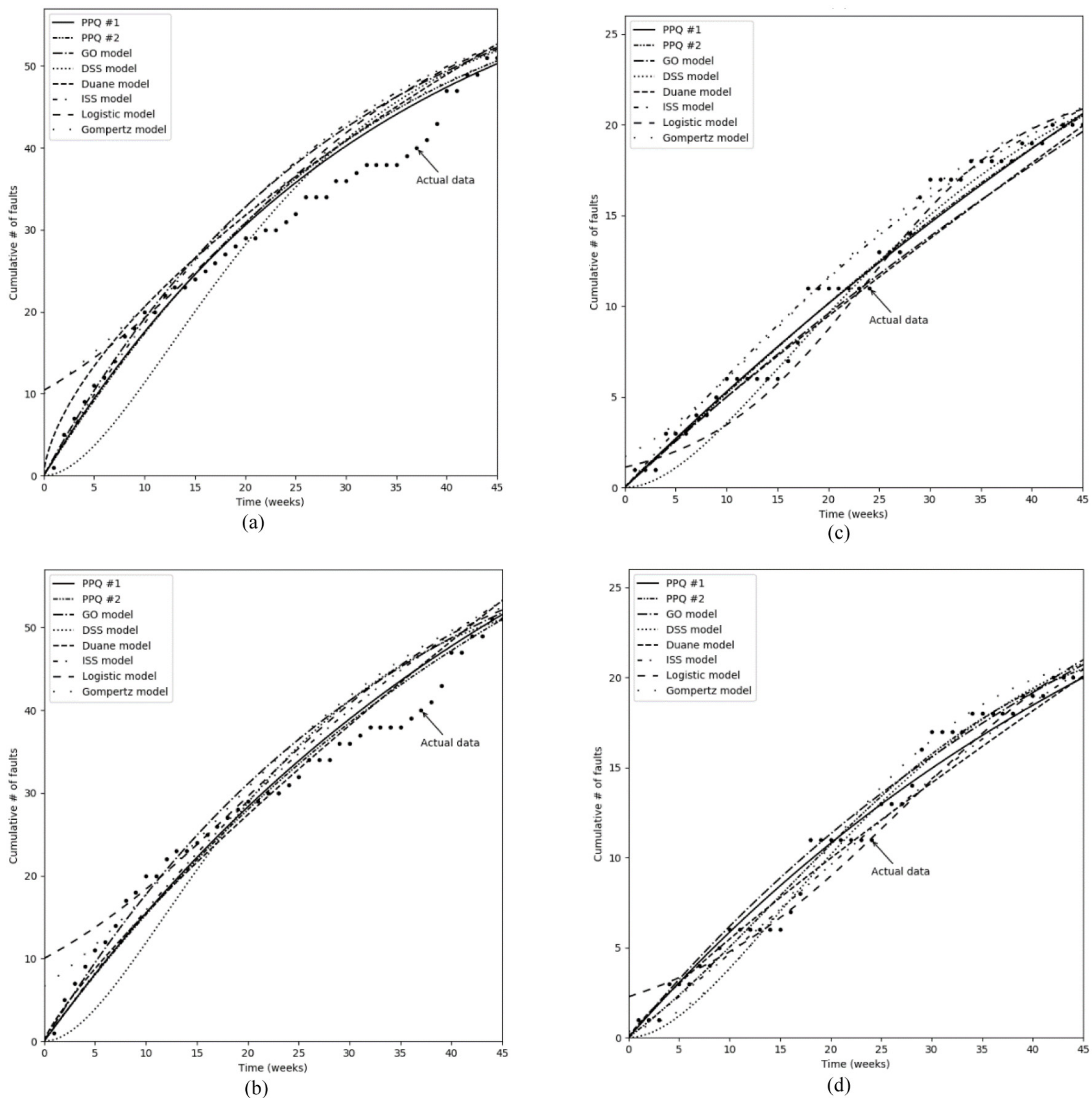

Fig. 5. Cumulative curves of actual and estimated number of removed faults (DS1): (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; (d) low priority using MLE.

Since $N_o(t)$ and $N_r(t)$ are independent of each other, we obtain

$$P\{N_{r,h}(t)=i\}P\{N_{r,l}(t)=j\}P\{N_o(t)=k\}$$
$$=\frac{[m_d(t)p_h]^i}{i!}e^{-m_d(t)p_h}\frac{[m_d(t)p_l]^j}{j!}e^{-m_d(t)p_l}\frac{[m_d(t)q]^k}{k!}e^{-m_d(t)q}. \tag{13}$$

Therefore, the number of removed faults in time t, $N_{r,h}(t)$ is a Poisson random variable with the mean value function:

$$m_{r,h}(t)=m_{d,h}(t)p_h=\int_0^t G(t-x)m'_{d,h}(x)dx. \tag{14}$$

Similarly, for the mean value function of low-priority faults $m_{d,l}(t)$ we have

$$m_{r,l}(t)=m_{d,l}(t)p_l$$
$$=\int_0^t(1-Q_0\sum_{n=c}^{\infty}\frac{(c\rho)^n}{c^{n-c}c!})G(t-x)m'_{d,l}(x)dx. \tag{15}$$

3.2. Performance and measurement of the PPQ model

There are some characteristics we are interested in (1) the average waiting time that a newly detected fault might be forced to wait and (2) the mean number of faults in the waiting queue and in the system. Since the behavior of queueing systems is a stochastic process, these measures are often calculated using random variables, probability distributions, or expected values (Shortle et al., 2018; Kleinrock, 2016). Here we denote the average fault detection rate as λ and the average fault removal rate as $c\mu$. Furthermore, there are c debuggers and each one is independent and identical with service rate μ. Here, we focus on the steady-state of the system in which ρ (traffic intensity) should be less than one.

As mentioned previously (see Section 3.1 above), there are two kinds of faults, high-priority and low-priority, with both following the Poisson process. Hence, we obtain

$$\rho=\frac{\lambda}{c\mu}=\rho_h+\rho_l=\frac{\lambda_h}{c\mu}+\frac{\lambda_l}{c\mu}, \tag{16}$$

where $\lambda=\lambda_h+\lambda_l$. Let $S_{h,k}$ denote the probability that the number of high-priority faults in the queueing system is k. We then obtain

$$S_{h,k}=P\{N_h=k\}. \tag{17}$$

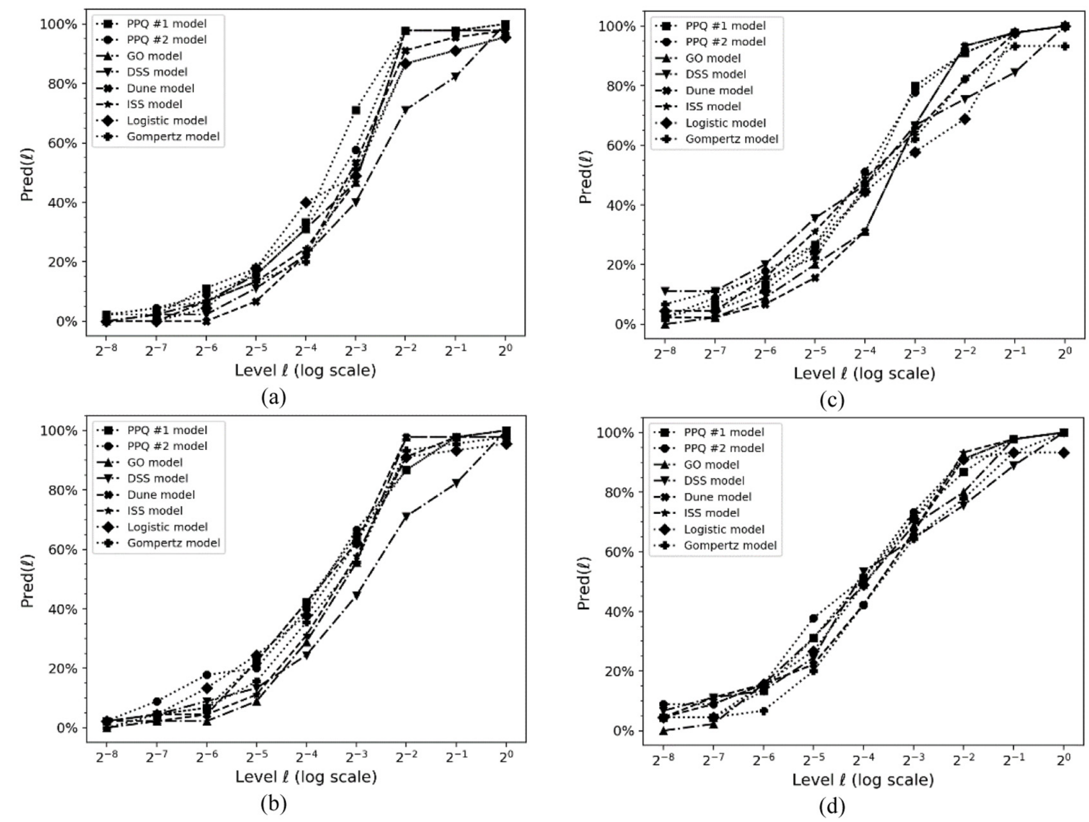

Fig. 6. The Pred(L) plot of selected models (log scale) for DS1: (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; (d) low priority using MLE.

Since the removal rate of faults depends on the number of debuggers in the system, if there are c or more faults being processed, then all debuggers should be busy. When there are fewer than c faults in the system ($k < c$), only k of the c debuggers are busy and the total removal rate of the system is $k\mu$. Therefore, we can obtain the removal rate at arbitrary time:

$$\mu_k = \begin{cases} k\mu, & \text{for } 1 \le k < c, \\ c\mu, & \text{for } k \ge c. \end{cases} \tag{18}$$

After substituting (16) and (18) into (17), it may be rewritten as

$$S_{h,k} = \begin{cases} S_{h,0}\dfrac{(c\rho_h)^k}{k!}, & \text{for } 1 \le k < c, \\ S_{h,0}\dfrac{c^c(\rho_h)^k}{c!}, & \text{for } k \ge c. \end{cases} \tag{19}$$

Thus, the expected number of high-priority faults in the steady state (including debugging) is given by

$$E[N_h] = \sum_{k=0}^{\infty} kS_{h,k} = c\rho_h + \frac{\rho_h}{1-\rho_h}C(c, c\rho_h), \tag{20}$$

where $C(c, c\rho_h)$ is referred to as *Erlang's c formula* (Shortle et al., 2018). In this study, we obtain

$$\begin{aligned} C(c, c\rho_h) &= \sum_{k=c}^{\infty} S_{h,k} = S_{h,c} + S_{h,c+1} + S_{h,c+2} + \cdots \\ &= S_{h,0}\frac{c^c(\rho_h)^c}{c!} + S_{h,0}\frac{c^c(\rho_h)^{c+1}}{c!} + S_{h,0}\frac{c^c(\rho_h)^{c+2}}{c!} + \cdots \\ &= S_{h,0}\frac{c^c(\rho_h)^c}{c!}\frac{1}{1-\rho_h}, \end{aligned} \tag{21}$$

as the probability that a newly detected high-priority fault is required to join the queue. Next, to get $S_{h,0}$, we use the fact that the probability $\{S_{h,k}\}$ must sum to 1:

$$\begin{aligned} 1 &= \sum_{k=0}^{\infty} S_{h,k} = [S_{h,0} + S_{h,1} + S_{h,2} + \cdots + S_{h,c-1}] + [S_{h,c} + S_{h,c+1} + \cdots] \\ &= [S_{h,0} + S_{h,0}\frac{(c\rho_h)}{1!} + S_{h,0}\frac{(c\rho_h)^2}{2!} + \cdots + S_{h,0}\frac{(c\rho_h)^{c-1}}{(c-1)!}] \\ &+ [S_{h,0}\frac{c^c(\rho_h)^c}{c!} + S_{h,0}\frac{c^c(c\rho_h)^{c+1}}{c!} + \cdots] \\ &= S_{h,0}[(1 + \frac{(c\rho_h)}{1!} + \frac{(c\rho_h)^2}{2!} + \cdots + \frac{(c\rho_h)^{c-1}}{(c-1)!}) + (\frac{c^c(\rho_h)^c}{c!})(\frac{1}{1-\rho_h})] \\ &= S_{h,0}[(\sum_{k=0}^{c-1}\frac{(c\rho_h)^k}{k!}) + \frac{c^c(\rho_h)^c}{c!(1-\rho_h)}]. \end{aligned} \tag{22}$$

Thus, according to Little's theorem and (20) we have

$$E[R_h] = \frac{c\rho_h}{\lambda_h} + \frac{\rho_h}{\lambda_h(1-\rho_h)}C(c, c\rho_h), \tag{23}$$

$$E[W_h] = \frac{\rho_h}{\lambda_h(1-\rho_h)}C(c, c\rho_h), \tag{24}$$

where $E[R_h]$ is the mean response time of high-priority faults (including waiting time and correcting time) and $E[W_h]$ is the mean waiting time of high-priority faults.

To estimate the performance in processing low-priority faults, the Conservation law (Kleinrock, 2016) will be applied. In our proposed model, no faults will be removed before debugging is completed or a new fault is introduced into the system. In other words, the remaining faults are independent of the order of

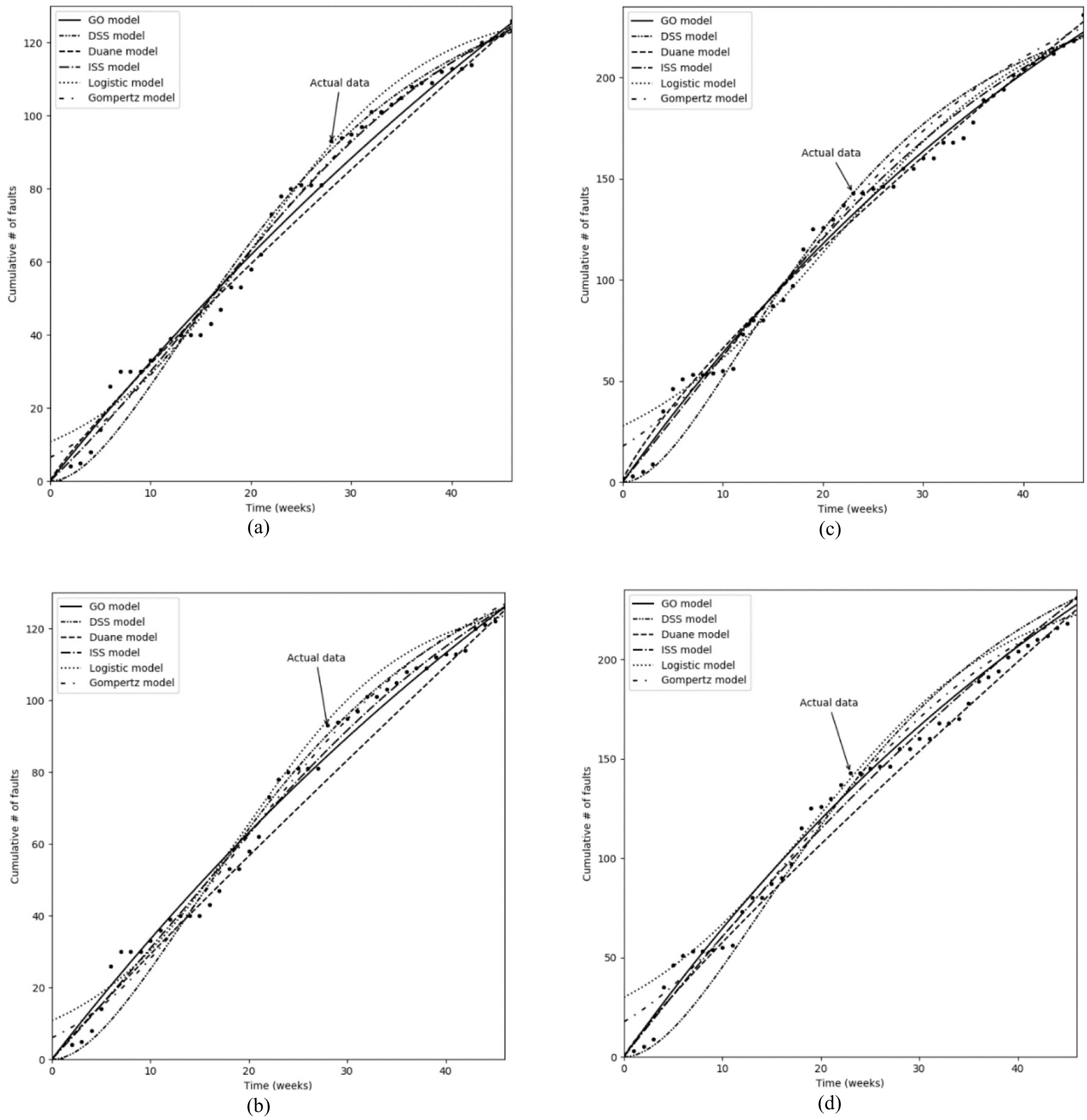

Fig. 7. Cumulative curves of actual and estimated number of detected faults (DS2): (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; (d) low priority using MLE.

debugging if the system is conservative. Thus, while preemptive scheduling will affect the mean waiting time of the individual priority levels, the average waiting time of the combined priority levels will be the same as under FCFS scheduling (Bondi and Buzen, 1984). Similarly, for the expected number of low-priority faults in the steady state we have

$$\begin{aligned} E[N_l] &= E[N] - E[N_h] \\ &= c\rho_l + \frac{\rho}{1-\rho}C(c, c\rho) - \frac{\rho_h}{1-\rho_h}C(c, c\rho_h), \end{aligned} \tag{25}$$

Where $E[N]$ is the total number of faults in the system. Hence, we also apply Little's theorem on (25) to obtain the following:

$$E[R_l] = \frac{c\rho_l}{\lambda_l} + \frac{\rho}{\lambda_l(1-\rho)}C(c, c\rho) - \frac{\rho_h}{\lambda_l(1-\rho_h)}C(c, c\rho_h), \tag{26}$$

and

$$E[W_l] = \frac{\rho}{\lambda_l(1-\rho)}C(c, c\rho) - \frac{\rho_h}{\lambda_l(1-\rho_h)}C(c, c\rho_h), \tag{27}$$

where $E[R_l]$ is the mean response time of low-priority faults (including waiting time and correcting time) and $E[W_l]$ is the mean waiting time of low-priority faults respectively.

4. Numerical examples

4.1. Selected datasets

In this paper, we have used three sets of failure data, including two datasets of OSS and one dataset of CSS. The characteristics of datasets are listed in Table 1. DS1 was collected from Bugzilla's (2016) public bug-tracking system, (Bugzilla, 2016). The system contains the shared components used by Firefox and other Mozilla software. DS2 was also obtained from Eclipse Bugzilla (2016). Eclipse is a large open source project that is an integrated development environment (IDE) used in computer programming. The project contains modeling tools, runtimes, reporting tools, and much more. It is supported by many developers around the world. For each report in Bugzilla, there are five priority levels denoted as P1, P2, P3, P4, and P5. Typically, P1 is the highest

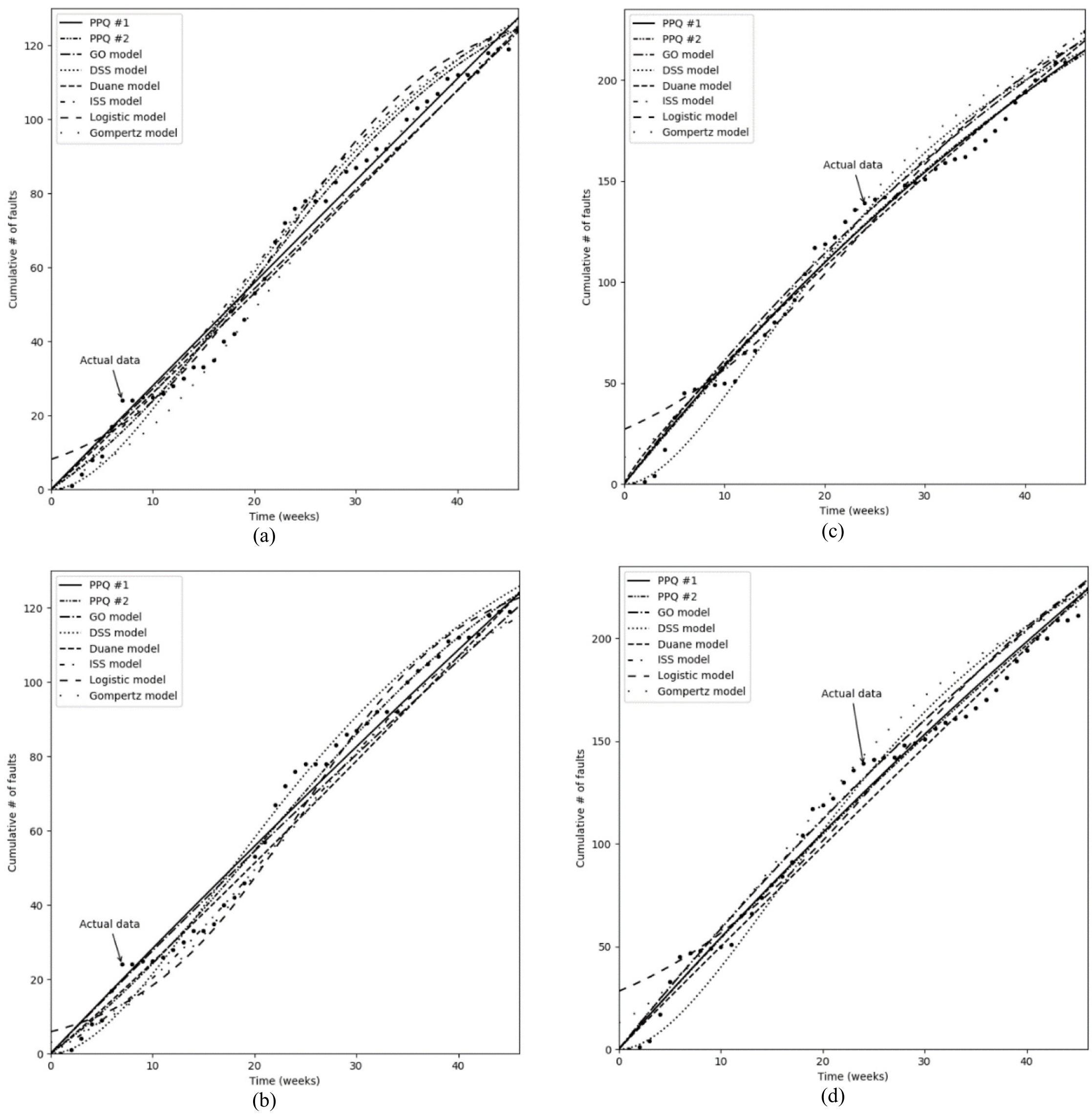

Fig. 8. Cumulative curves of the actual and estimated number of removed faults (DS2): (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; (d) low priority using MLE.

Table 1
Summary of failure datasets.

Datasets	Duration	Failures count	Remarks
DS1 (Bugzilla, 2016)	311 Days	81	The failure data was collected from the open source software project Mozilla Core component from 2013/1/2 to 2013/11/9.
DS2 (Eclipse Bugzilla, 2016)	318 Days	357	The failure data was collected from the open source software project Eclipse JDT core version 2.0 from 2002/1/2 to 2002/11/16.
DS3 (CSS)	192 Days	103	The failure data was obtained from the bug-tracking system used by Coretronic Corporation in Taiwan from 2014/02/04 to 2014/08/15.

priority. The software system can usually be released only if all P1 faults are corrected. On the other hand, P5 is the lowest priority and faults assigned this priority might remain unfixed for a long period of time. To meet the usage required of our proposed model, we will divide five priority levels into two types: high-priority (P1 and P2), and low-priority (P3, P4 and P5). It is also noted that since the development of OSS is usually accomplished by volunteers (Samoladas et al., 2010), only resolved reports with the status value "resolved", "closed", or "confirmed" with "fixed" and "duplicate" resolution have been included in our datasets. Thus, the overall failure datasets fall into two types: time domain data and interval domain data (Musa et al., 1987). DS3 was from a

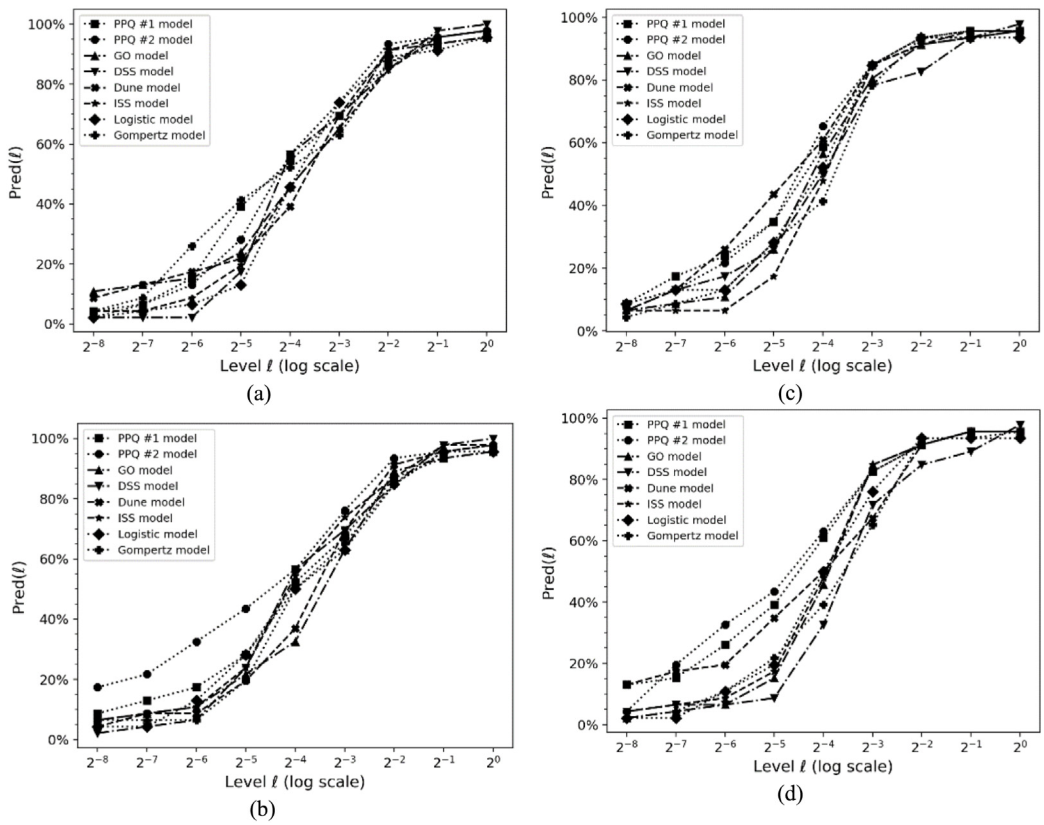

Fig. 9. The Pred(L) plot of selected models (log scale) for DS2: (a) high priority using LSE; (b) high priority using MLE; (c) low priority using LSE; and (d) low priority using MLE.

projector firmware project developed by Coretronic Corporation in Taiwan. During a 28-week software testing process, 99 faults were removed. It is also noticed that DS1–DS3 were collected in interval domain format. Original data for DS1, DS2, and DS3 are given in Appendix A, B, and C, respectively.

It is worth noting that, if the failure rates decrease with time, it may mean the software system has become stable and its reliability has grown. Conversely, incremental growth in the failure rate may imply a decay in software reliability. To determine whether the software underwent reliability growth or decline, Kanoun and Laprie (1994) proposed using the Laplace trend test on failure data. The trend analyzes allow periods of reliability growth and reliability decline to be identified for the application of SRGMs to data exhibiting trends under the modeling assumptions. Here we plan to use Laplace trend analysis to determine whether the software has undergone reliability growth or decline.

In the various analytical tests, the Laplace test is the most widely used in identifying trends in group or time-series data. We calculate the Laplace trend factor $U(t)$ as follows. Let time interval $(0, t]$ be divided into t equal units of time, and let x_i be the number of faults detected during time unit i. Then, $U(t)$ is defined as (Lyu, 1996)

$$U(t) = \frac{\sum_{i=1}^{t}(i-1)x_i - \frac{t-1}{2}\sum_{i=1}^{t}x_i}{\sqrt{\frac{t_2-1}{12}\sum_{i=1}^{t}x_i}}. \quad (28)$$

Negative values of $U(t)$ indicate a decreasing failure intensity (reliability growth), while positive values indicate an increasing failure intensity (reliability decline). Values varying between −2 and +2 represent stable reliability. Using DS1 and DS2, Fig. 3 shows the Laplace trend test results. It is noted that Fig. 3(a) shows four regions of Laplace trend change for DS1: [1, 24], [25, 28], [29, 36], and [37, 45]. The first region depicts a continued downward trend between the 1st week and the 24th week. The decreasing failure intensity indicates a growth in reliability. In the second region, the value of $U(t)$ increases slightly from the 25th week to the 28th week, which means that the reliability falls. The third region, between the 29th week and the 36th week, shows that the failure intensity is declining, indicating a steady increase in reliability. In the last region, the value of $U(t)$ significantly rebounded from the 37th to the 45th week, presenting a downward trend in reliability.

Similarly, Fig. 3(b) shows that DS2 may be divided into four regions: [1, 11], [12, 23], [24, 44], and [45, 46]. In the first region, there was a large change in the value of $U(t)$ between the 1st week and the 11th week, which obviously indicates a growth in reliability. However, the second region shows a significant increase in the failure intensity from the 12th week to the 23th week, representing a decline in software reliability. In the period from the 24th week to the 44th week, the third region demonstrates a downward trend in failure intensity, indicating a growth in reliability. In the last region, between the 45th and the 46th weeks, there is a small increase in the value of $U(t)$, meaning a slight decrease in the reliability of the software.

Fig. 3(c) shows four regions of Laplace trend change for DS3: [1, 5], [6, 12], [13, 19], and [20, 28]. The first region, between the 1st and the 5th weeks, shows an obvious increase in the failure intensity, representing decreasing reliability. However, it can be seen that the second region shows a significant decrease in the failure intensity from the 6th week to the 12th week, representing the growth in software reliability. In the third region, from the 13th week to the 19th week, the value of $U(t)$ grows significantly,

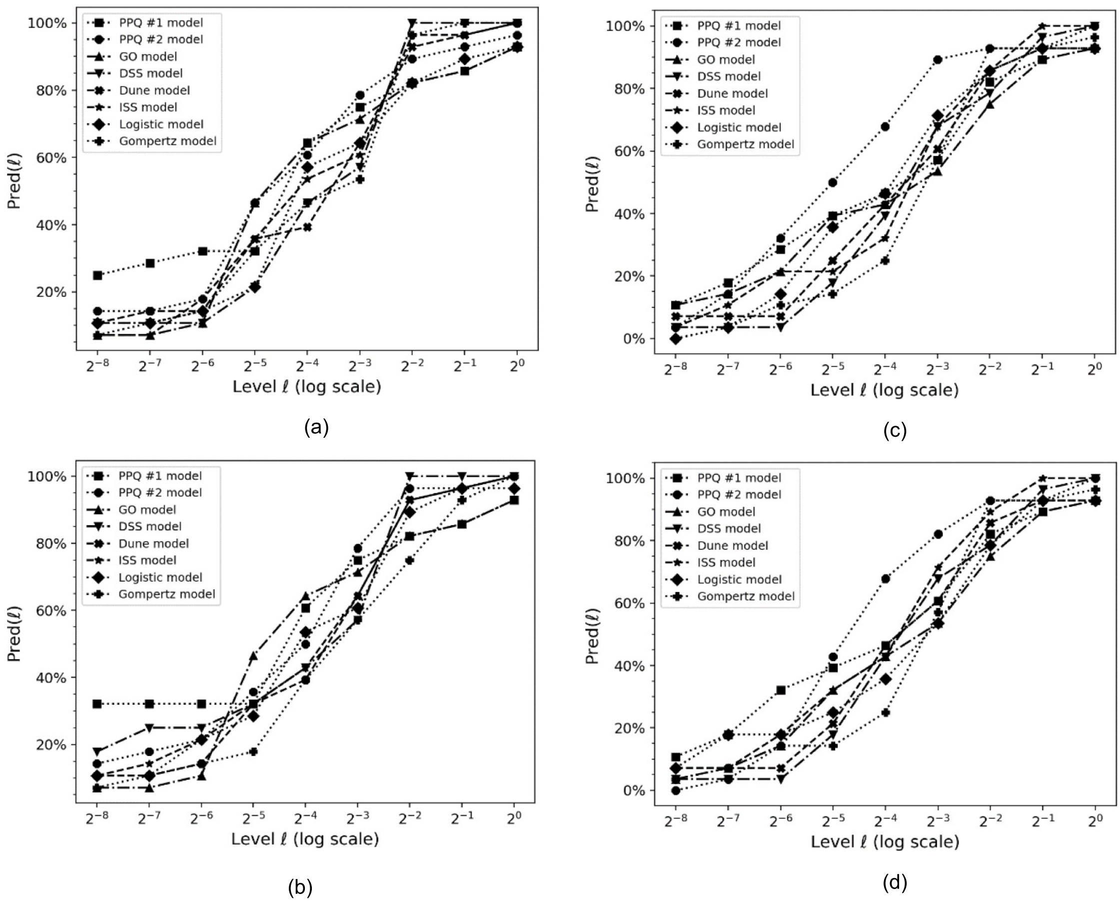

Fig. 10. The Pred(L) plot of selected models (log scale) for DS3: (a) high-priority by LSE; (b) high-priority by MLE; (c) low-priority by LSE; (d) low-priority by MLE.

Table 2
Classification scheme of selected models.

Models (High-priority)	Descriptions
Duane model (Lyu, 1996)	$m_d(t) = m_r(t)$
GO model (Musa et al., 1987; Xie, 1991)	$m_d(t) = m_r(t)$
DSS model (Lyu, 1996; Xie, 1991)	$m_d(t) = m_r(t)$
ISS model (Lyu, 1996; Xie, 1991)	$m_d(t) = m_r(t)$
Logistic model (Xie, 1991)	$m_d(t) = m_r(t)$
Gompertz model (Xie, 1991)	$m_d(t) = m_r(t)$
Proposed PPQ model #1	$m_d(t)$: **GO**; $m_r(t)$: (14)
Proposed PPQ model #2	$m_d(t)$: **ISS**; $m_r(t)$: (14)
Models (Low-priority)	**Descriptions**
Duane model (Lyu, 1996)	$m_d(t) = m_r(t)$
GO model (Musa et al., 1987; Xie, 1991)	$m_d(t) = m_r(t)$
DSS model (Lyu, 1996; Xie, 1991)	$m_d(t) = m_r(t)$
ISS model (Lyu, 1996; Xie, 1991)	$m_d(t) = m_r(t)$
Logistic model (Xie, 1991)	$m_d(t) = m_r(t)$
Gompertz model (Xie, 1991)	$m_d(t) = m_r(t)$
Proposed PPQ model #1	$m_d(t)$: **GO**; $m_r(t)$: (15)
Proposed PPQ model #2	$m_d(t)$: **ISS**; $m_r(t)$: (15)

showing that reliability is declining. In the last region, there is a downward trend in failure intensity, representing stable growth in reliability.

Overall, as shown in Fig. 3, there are indications of reliability growth after the end of testing. In this case, our proposed model and selected SRGMs can be applied to predict the number of detected faults for DS1, DS2, and DS3.

4.2. Criteria for model comparison

In this paper, we employed some criteria to evaluate and compare the performance of our proposed model and other selected SRGMs (Kanoun et al., 1991).

(1) The *Mean Absolute Error* (MAE) is defined by Sukhwani et al. (2016), Chai and Draxler (2014):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|m_i - m(t_i)|. \tag{29}$$

where m_i is the actual number of detected or removed faults by time t_i, and $m(t_i)$ is the expected number of faults by time t_i.
The MAE indicates the magnitude of the average error. It is a linear score, which means that all the individual differences are weighted equally on average.

(2) The *Mean Square Error* (MSE) is used for long-term prediction and is typically defined as Huang and Kuo (2017), Sukhwani et al. (2016), Conte et al. (1986):

$$MSE = \frac{1}{n-\theta}\sum_{i=1}^{n}(m_i - m(t_i))^2, \tag{30}$$

where n is the size of the selected dataset and θ is the degree of freedom. A smaller MSE means less fitting error and better performance.

(3) *Theil's U Statistic* (TS) is a widely used criterion for evaluating the estimated results. The TS includes two parts,

Table 3
Parameter estimation results of models for detected faults (DS1).

LSE				
Models (High-priority)	a	b	ψ	k
Duane model	–	0.597	–	0.055
GO model	64.82	0.039	–	–
DSS model	57.20	0.090	–	–
ISS model	62.92	0.047	0.334	–
Logistic model	58.78	0.085		3.918
Gompertz model	59.14	0.938		0.137
Models (Low-priority)	a	b	ψ	k
Duane model	–	0.746	–	0.615
GO model	37.92	0.023	–	–
DSS model	30.60	0.068	–	–
ISS model	29.47	0.049	0.739	–
Logistic model	25.87	0.106	–	7.952
Gompertz model	26.49	0.924	–	0.049
MLE				
Models (High-priority)	a	b	ψ	k
Duane model	–	0.654	–	0.101
GO model	86.42	0.023	–	–
DSS model	59.49	0.091	–	–
ISS model	88.25	0.023	0.013	–
Logistic model	56.34	0.091	–	4.059
Gompertz model	66.38	0.950	–	0.159
Models (Low-priority)	a	b	ψ	k
Duane model	–	0.845	–	0.997
GO model	65.36	0.011	–	–
DSS model	29.63	0.074	–	–
ISS model	65.01	0.011	0.013	–
Logistic model	28.38	0.093	–	9.891
Gompertz model	26.67	0.924	–	0.0269

denoted as U1 and U2. A low value of U1 and U2 indicates a more accurate predictive capability of the model. *Theil's U Statistics* are defined by Huang and Kuo (2017), Li et al. (2005), Holden et al. (1991):

$$U1 = \frac{\sqrt{\sum_{i=1}^{n}(m_i - m(t_i))^2}}{\sqrt{\sum_{i=1}^{n} m_i{}^2} + \sqrt{\sum_{i=1}^{n} m(t_i)^2}}, \tag{31}$$

and

$$U2 = \frac{\sqrt{\sum_{i=1}^{n-1}(\frac{m(t_{i+1})-m_{i+1}}{m_i})^2}}{\sqrt{\sum_{i=1}^{n-1}(\frac{m_{i+1}-m_i}{m_i})^2}}. \tag{32}$$

(4) The *Coefficient of Determination* (R^2) measures the percentage of variations in the model. The value of R^2 is between 0 and 1, and a higher R^2 value indicates better fitness of the model. The R^2 value is defined as Conte et al. (1986), Keller and Warrack (1999):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(m(t_i) - m_i)^2}{\sum_{i=1}^{n}(m_i - \overline{m})^2}, \tag{33}$$

and

$$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{m_i}. \tag{34}$$

(5) *Variance* is a measure of how far a set of numbers is spread out. It is defined as Huang and Kuo (2017), Pillai and Nair (1997):

$$Variance = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(m_i - m(t_i) - Bias)^2}, \tag{35}$$

and

$$Bias = \sum_{i=1}^{n}\frac{m(t_i) - m_i}{n}. \tag{36}$$

Prediction *Bias* is the average of the prediction errors.

(6) To measure the relative quality of the models we applied to the failure data, the *Akaike Information Criterion* (*AIC*) was used to provide a standard for model selection. The *AIC* is defined as Hsu and Huang (2014), Sukhwani et al. (2016), Akaike (1974):

$$AIC = 2\theta - 2ln(L), \tag{37}$$

where L is the maximized value of the likelihood function of the model. In a set of candidate models for the target data, the one with the lowest *AIC* value is the preferred model.

(7) The *prediction at level l* is defined as Fenton and Pfleeger (1998), Shin and Goel (2000)

$$\mathrm{Pred}(l) = \frac{k}{n}, \tag{38}$$

where k indicates the number of n data points whose magnitude of relative error is less than or equal to l. Typically l is set to 0.25, thus Pred(0.25) indicates the percentage of estimated values that are within 25% of the actual data. The larger value of Pred(l) on a fixed l means a more precise estimation of the model.

Table 4
Performance comparisons of models for detected faults (DS1).

LSE							
Models (High-priority)	MAE	*MSE*	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	2.586	9.879	0.044	1.072	0.956	5.422	––
GO model	3.863	20.057	0.062	1.080	0.869	6.857	––
DSS model	4.625	30.388	0.080	2.674	0.801	5.985	––
ISS model	3.462	15.631	0.055	1.201	0.897	5.179	––
Logistic model	3.726	20.408	0.063	1.929	0.866	7.879	––
Gompertz model	3.066	13.405	0.051	1.168	0.912	5.764	––
Models (Low-priority)	MAE	*MSE*	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	0.866	1.263	0.035	1.308	0.972	1.477	––
GO model	1.057	1.704	0.041	0.755	0.962	1.997	––
DSS model	1.635	3.892	0.063	1.798	0.913	2.133	––
ISS model	1.161	1.920	0.043	0.721	0.957	2.196	––
Logistic model	1.158	2.056	0.045	2.035	0.954	1.875	––
Gompertz model	1.305	2.541	0.049	0.932	0.943	2.249	––
MLE							
Models (High-priority)	MAE	*MSE*	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	1.893	6.018	0.035	0.865	0.961	3.208	82.97
GO model	3.272	14.394	0.054	1.424	0.906	3.967	121.33
DSS model	5.203	35.626	0.084	2.658	0.766	6.038	161.22
ISS model	3.423	15.578	0.056	1.440	0.898	4.186	124.82
Logistic model	3.292	15.945	0.056	1.683	0.895	6.617	125.84
Gompertz model	3.731	19.749	0.061	1.682	0.871	7.879	135.26
Models (Low-priority)	MAE	*MSE*	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	0.898	1.356	0.038	0.767	0.968	1.181	17.70
GO model	0.932	1.493	0.040	0.642	0.965	1.236	22.03
DSS model	1.579	3.512	0.060	1.686	0.918	1.895	60.53
ISS model	0.947	1.583	0.041	0.637	0.963	1.288	24.67
Logistic model	1.510	3.136	0.057	1.739	0.930	1.907	53.40
Gompertz model	1.319	2.435	0.051	1.109	0.943	1.598	44.05

In addition to our proposed PPQ models, six SRGMs will also be used for comparing the prediction performance of models on the selected dataset. Table 2 shows a summary of selected candidate models.

4.3. Case study I—DS1

First, we will analyze the FDP of DS1. Six SRGMs will be applied to fit against the real failure data. It is worth noting that all faults were classified into two types, high and low-priority. The methods of Least Square Estimation (LSE) Maximum Likelihood Estimation (MLE) are used to estimate the parameters of all selected models (Musa et al., 1987; Lyu, 1996; Xie, 1991). The results of the parameter estimation for all models are listed in Table 3. The estimated number of detected faults versus time is shown in Fig. 4. Figs. 4(a) and 4(b) show that there are significant differences between the actual data and fitted curves of all selected models between the 25th and the 40th weeks using the LSE and MLE methods. This is likely because large numbers of high priority faults (about 64%) were detected from the 1st to the 25th weeks, but the average high priority fault detection rate decreased significantly from the 26th to the 45th weeks. The difference in the high priority fault detection rate between the early and final periods leads to inaccurate prediction by the candidate models. Another possible reason is that most newly detected faults were low priority in the period from the 25th to the 40th weeks, leading to a difference between the predicted and actual data. Traditional SRGMs do not take into account the priority level, and regard the different priority faults as identical. Thus, this phenomenon may occur when we discuss different priorities separately. Note that the curves in Fig. 4 will be very close to the actual number of detected faults at the end of detection in the 45th week.

Table 4 shows the results of the performance comparisons of all selected models for detected faults. Note that the Duane model has the lowest values for MAE, MSE, TS-U1 and AIC in the process of fault detection. As we can see from Table 5, the GO and ISS models also provide lower values for all criteria, which means that the GO and ISS models are second and third, respectively, behind the Duane model.

Similarly, we estimate the parameters of all selected models using LSE and MLE during the FCP. Because we cannot find any information about the precise number of debuggers for DS1, here we use the number of core contributors to estimate the number of debuggers (parameter c) (Makaveli, 2017). For instance, DS1 was collected from the Mozilla Core component. Based on the analysis of the failure data, more than 50% of faults are corrected by the top 20 core contributors (Ye and Kishida, 2003). Furthermore, the total number of detected faults decreased during our data collection. Britton et al. (2013) found that software developers spend 50% of their programming time fixing bugs and making code work. Therefore, we can reasonably assume that the total number of debuggers c is 10 persons.

Table 5 shows the results of the parameter estimations of the models for corrected faults. It can be found that the parameter b of the Duane model is less than 1 in both LSE and MLE. According

Table 5
Parameter estimations of models for corrected faults (DS1).

LSE							
Models (High-priority)	*a*	*b*	*c*	*k*	ψ	μ	λ
Duane model	–	0.624	–	0.078	–	–	–
GO model	66.45	0.031	–	–	–	–	–
DSS model	59.31	0.078	–	–	–	–	–
ISS model	63.32	0.045	–	–	0.384	–	–
Logistic model	58.26	0.082	–	4.599	–	–	–
Gompertz model	60.55	0.950	–	0.165	–	–	–
Proposed PPQ model #1	68.14	0.030	10	–	–	0.148	1.162
Proposed PPQ model #2	62.79	0.042	10	–	0.385	0.183	1.493
Models (Low-priority)	*a*	*b*	*c*	*k*	ψ	μ	λ
Duane model	–	0.921	–	1.748	–	–	–
GO model	65.22	0.008	–	–	–	–	–
DSS model	26.42	0.064	–	–	–	–	–
ISS model	27.64	0.042	–	–	0.843	–	–
Logistic model	22.27	0.125	–	19.04	–	–	–
Gompertz model	21.32	0.938	–	0.069	–	–	–
Proposed PPQ model #1	65.14	0.008	10	–	–	0.079	0.704
Proposed PPQ model #2	34.76	0.030	10	–	1.002	0.051	0.423
MLE							
Models (High-priority)	*a*	*b*	*c*	*k*	ψ	μ	λ
Duane model	–	0.821	–	0.355	–	–	–
GO model	71.94	0.283	–	–	–	–	–
DSS model	57.08	0.085	–	–	–	–	–
ISS model	95.14	0.018	–	–	0.010	–	–
Logistic model	59.30	0.080	–	4.931	–	–	–
Gompertz model	61.95	0.944	–	0.107	–	–	–
Proposed PPQ model #1	90.27	0.019	10	–	–	0.135	1.181
Proposed PPQ model #2	88.97	0.019	10	–	0.013	0.129	1.255
Models (Low-priority)	*a*	*b*	*c*	*k*	ψ	μ	λ
Duane model	–	0.865	–	1.402	–	–	–
GO model	36.91	0.018	–	–	–	–	–
DSS model	25.62	0.068	–	–	–	–	–
ISS model	30.94	0.043	–	–	2.013	–	–
Logistic model	25.54	0.086	–	10.25	–	–	–
Gompertz model	22.82	0.918	–	0.015	–	–	–
Proposed PPQ model #1	37.32	0.017	10	–	–	0.059	0.524
Proposed PPQ model #2	23.65	0.073	10	–	3.186	0.068	0.620

to the definition of the Duane model, if parameter b is less than 1, the system exhibits a reliability growth trend (Ho and Xie, 1998). The Laplace trend test results in Fig. 3(a) also show that reliability is increasing. For the ISS model and PPQ model #2, we estimate the inflection parameter (ψ), which is defined as $\psi = (1\text{-}\varphi)/\varphi$ where φ is the inflection rate [3]. The inflection rate represents the ratio of the number of detectable faults to the total number of faults in the program. For the high-priority faults of DS1, the estimated value ψ of the ISS model by LSE is 0.384. We then obtain $\varphi = 0.722$ which means most of the faults are detectable from the beginning of testing. For low-priority faults, the ψ of PPQ model #2 by MLE is 3.186, giving an inflection rate of 0.238, which indicates that only a few faults are detectable at the beginning of testing.

Fig. 5 shows the cumulative amount of the actual and estimated number of corrected faults versus time in DS1. In Figs. 5(a) and 5(b), the estimation results are significantly different from the actual data for the 25th through the 40th week. This is likely because the number of open-remaining high-priority faults is close to zero. Consequently, the actual cumulative number of removed high priority faults grows slowly from the 25th to the 40th week.

Table 6 shows the comparison results of the selected models for corrected faults in DS1. As Table 6 makes clear, PPQ model #2 has the lowest values for MAE, MSE, TS-U1, and AIC of all SRGMs in the process of fault correction. The PPQ model #2 also provides the largest R2 value, which means a better fit to DS1. Moreover, PPQ model #1 has the lowest values for MAE, MSE, TS-U1, TS-U2, and AIC among all SRGMs, which means the PPQ model #1 in the fault correction estimation is second only to PPQ model #2.

Finally, Fig. 6 depicts the Pred(l) plots of the selected models for different l levels ranging from 2–8 to 1. In general, an estimation model with a higher value of Pred(0.25) provides more accurate estimates than a model with a lower value. In Fig. 6(a), the Pred(0.25) values for the Duane model, the GO model, the DSS model, the ISS model, the Logistic model, the Gompertz model, and the proposed PPQ models #1 and #2 are 0.91 0.97, 0.71, 0.97, 0.86, 0.86, 0.97, and 0.97, respectively. Thus, 97% of the estimates from the GO model, the ISS model, and PPQ models #1 and #2, are within 25% of the actual data. Overall, the Pred(l) plots show that PPQ models #1 and #2 perform well at different l levels in Fig. 6. Finally, an examination of Table 6 suggests that on average, PPQ models #1 and #2 provide a better fit for DS1 and predict future FCPs well.

4.4. Case study II—DS2

Similarly, the parameters of all selected models for high- and low-priority faults are estimated by the methods of LSE and MLE

Table 6
Performance comparisons of models for corrected faults (DS1).

LSE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	3.381	14.976	0.056	0.680	0.911	7.051	–
GO model	3.592	20.687	0.065	0.793	0.877	7.429	–
DSS model	4.478	27.513	0.078	1.206	0.836	5.194	–
ISS model	3.841	23.170	0.069	0.802	0.862	7.658	–
Logistic model	3.704	21.487	0.067	1.741	0.872	7.831	–
Gompertz model	3.681	19.088	0.063	1.758	0.886	7.684	–
Proposed model #1	2.425	9.151	0.045	0.515	0.946	4.010	–
Proposed model #2	2.733	10.579	0.050	0.562	0.931	4.586	–
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	1.085	1.976	0.055	1.626	0.951	2.091	–
GO model	1.085	1.911	0.539	1.632	0.953	2.062	–
DSS model	0.928	1.488	0.046	1.364	0.963	1.772	–
ISS model	1.308	2.545	0.057	1.781	0.937	2.712	–
Logistic model	1.001	1.671	0.049	0.912	0.959	1.639	–
Gompertz model	1.008	1.644	0.047	1.148	0.959	2.099	–
Proposed model #1	0.749	0.967	0.037	0.586	0.976	1.113	–
Proposed model #2	0.720	0.893	0.036	0.594	0.978	1.074	–
MLE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	2.704	10.792	0.048	0.801	0.900	3.863	109.82
GO model	2.932	13.421	0.053	0.847	0.920	5.302	118.81
DSS model	4.245	25.217	0.074	1.185	0.850	5.964	147.19
ISS model	2.917	12.136	0.052	0.927	0.924	3.990	114.28
Logistic model	3.099	16.193	0.059	0.961	0.904	6.290	127.27
Gompertz model	3.331	17.325	0.060	1.086	0.897	6.477	130.30
Proposed model #1	2.413	8.453	0.043	0.678	0.949	2.932	98.01
Proposed model #2	2.273	7.668	0.042	0.689	0.954	2.738	93.62
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	0.919	1.428	0.046	0.633	0.965	1.465	37.99
GO model	0.842	1.217	0.042	0.716	0.970	1.402	32.80
DSS model	0.856	1.744	0.049	0.845	0.971	1.384	38.02
ISS model	2.287	1.201	0.094	1.061	0.813	4.736	30.21
Logistic model	0.927	1.538	0.047	1.228	0.962	1.486	31.32
Gompertz model	0.962	1.345	0.043	1.070	0.967	1.168	35.34
Proposed model #1	0.815	1.077	0.039	0.636	0.973	1.026	27.39
Proposed model #2	0.657	0.700	0.031	0.664	0.983	0.853	25.17

for DS2. The results of the parameter estimation for all selected models are listed in Table 7.

Note that the Eclipse project began to develop as open source software in November 2001. DS2 was collected for the period from January 2, 2002 to November 16, 2002, the early stage of the software development process. In the early stage, there are many faults reported to the bug-tracking system, and many developers are devoted to the project to meet its human resource needs. Hence, we set the total number of debuggers for DS2 at 40 persons. We list the performance comparison of all models for detected faults in Table 8. The GO and ISS model have the best results for MAE, MSE, TS-U1, TS-U2, variance, and AIC among selected models. Thus, both exhibit a good fit to the actual failure data of DS2.

The estimated expected number of detected faults for each model is shown in Fig. 7, which shows that the cumulative number of detected faults (both high priority and low priority) increases stably. The results of the estimation for FDP correspond well with the actual failure data. All the curves will be very close to the actual number of detected faults at the end of detection in the 46th week.

Table 9 gives the estimation results of the parameters of all models for corrected faults for DS2. In Table 9, we observe that the value of parameter b for high-priority faults of the Duane model for the LSE and MLE methods is estimated to be 1.026 and 1.061, respectively. Based on the definition of the Duane model, if parameter b is greater than 1, system reliability is declining (Ho and Xie, 1998). This result is consistent with the fact that DS2 is in the early stages of the software development process,

Table 7
Parameter estimation results of models for detected faults (DS2).

LSE				
Models (High-priority)	a	b	ψ	k
Duane model	–	0.891	–	0.204
GO model	331.86	0.010	–	——
DSS model	140.59	0.078	–	——
ISS model	145.30	0.067	2.695	——
Logistic model	130.05	0.118	–	11.152
Gompertz model	142.10	0.936	–	0.045
Models (Low-priority)	a	b	ψ	k
Duane model	–	0.811	–	0.057
GO model	414.03	0.017	–	——
DSS model	242.80	0.086	–	
ISS model	272.28	0.052	1.296	——
Logistic model	240.10	0.096	–	7.672
Gompertz model	260.82	0.939	–	0.068
MLE				
Models (High-priority)	a	b	ψ	k
Duane model	–	0.952	–	0.288
GO model	301.48	0.012	–	——
DSS model	147.67	0.739	–	——
ISS model	163.98	0.051	1.879	——
Logistic model	128.55	0.122	–	10.952
Gompertz model	148.51	0.937	–	0.040
Models (Low-priority)	a	b	ψ	k
Duane model	–	0.894	–	0.107
GO model	438.03	0.016	–	——
DSS model	272.74	0.073	–	——
ISS model	582.25	0.011	0.013	——
Logistic model	238.10	0.100	–	6.972
Gompertz model	263.82	0.941	–	0.067

in which a large number of faults are detected. Moreover, for the high-priority faults of DS2, it can be found from Table 9 that the estimated value of the inflection factor for the ISS model by the method of LSE is 5.034. Thus, we obtain the inflection rate $\varphi = 0.165$, which means that only a few faults are detectable at the beginning of testing. Using the low-priority faults in DS2, the value of parameter ψ of PPQ model #2 by MLE is 0.135 and the value of inflection rate φ is 0.881, indicating that most faults in the software are detectable at the beginning of testing.

Fig. 8 illustrates the cumulative number of the actual and estimated number of removed faults versus time in DS2. The estimated results appear to fit well to the actual data. Table 10 shows the comparison criteria results for the six selected models in DS2. PPQ models #1 and #2 have values for MAE, MSE, TS-U1, and AIC smaller than those of the traditional models.

Finally, Fig. 9 shows the Pred(l) plots of the selected models for different l levels ranging from 2^{-8} to 1. In Fig. 9(b), the Pred(0.25) values for the Duane model, the GO model, the DSS model, the ISS model, the Logistic model, the Gompertz model, and the proposed PPQ models #1 and #2 are 0.91 0.89, 0.84, 0.86, 0.84, 0.84, 0.87, and 0.93, respectively. Results show that 93% of the estimates from PPQ #2 (the best model), are within 25% of the actual data. Overall, the Pred(l) plots show that PPQ models #1 and #2 perform well at different l levels in Fig. 9. Altogether, it can be found that PPQ models #1 and #2 are capable of providing reliability prediction with acceptable accuracy for this dataset.

4.5. Case study III—DS3

Similarly, the estimated values of the parameters of all selected models by using the methods of LSE and MLE are listed in Table 11. Here, it is noted that during the time interval we collected the failure data, the functionality of the projector system was still being modified to meet customer needs. For DS3, we collected failure data from the development stage of the software development process. The failure report recorded seven debuggers involved in the projector firmware system. Table 12 gives the comparison results of all models for detected faults in DS3. We can see that the GO and ISS models have the lowest results for MAE, MSE, TS-U1, and AIC among the models for the fault detection process, which means that both predict the future fault detection accurately for the DS3 dataset.

Table 13 shows the estimated values of the model's parameters for corrected faults of DS3. We can see that the value of parameter b of the Duane model for the high- and low-priority faults is 1.236 and 1.029, respectively. In general, parameter b greater than 1 indicates that the system is deteriorating. This is consistent with the result of the Laplace trend test in Fig. 3(c) indicating a negative growth in reliability. Moreover, as shown in Table 13, the estimated values of parameter ψ for the ISS model and PPQ model #2 according to the MLE are 8.28 and 5.72, respectively. Thus, for the ISS model and PPQ model #2, the inflection rate φ is 0.126 and 0.148, respectively. They both indicate that only a few faults are detectable at the beginning of testing.

Table 14 also gives the results of the performance comparison of all models for corrected faults of DS3. We can see that PPQ model #2 has better results for MAE, MSE, TS-U1, and AIC compared to other traditional SRGMs. Moreover, PPQ model #1 also provides lower values for MAE, MSE, TS-U1, TS-U2, and AIC than the traditional models, which means that PPQ model #1 is second only to PPQ model #2 in the fault correction process.

Table 8
Performance comparisons of models for detected faults (DS2).

LSE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	4.899	35.905	0.039	1.530	0.975	7.690	–
GO model	4.009	22.820	0.030	1.346	0.984	5.164	–
DSS model	3.918	28.192	0.033	1.222	0.980	5.370	–
ISS model	3.126	15.303	0.025	0.991	0.989	3.958	–
Logistic model	4.486	26.192	0.032	2.240	0.982	6.726	–
Gompertz model	3.386	17.011	0.026	1.517	0.988	4.173	–
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	5.710	50.359	0.025	1.728	0.988	7.224	–
GO model	5.423	44.342	0.023	1.224	0.989	6.734	–
DSS model	9.384	152.62	0.043	0.961	0.963	12.872	–
ISS model	6.175	53.294	0.026	1.185	0.986	8.172	–
Logistic model	7.583	105.87	0.036	3.428	0.974	11.104	–
Gompertz model	6.614	66.912	0.029	2.290	0.984	8.289	–
MLE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	5.647	49.708	0.046	1.205	0.965	10.692	182.68
GO model	3.817	20.790	0.029	1.433	0.985	4.610	143.58
DSS model	4.318	31.030	0.035	1.297	0.978	5.632	162.01
ISS model	3.365	16.165	0.025	1.097	0.989	4.066	132.01
Logistic model	5.116	34.200	0.036	2.306	0.976	8.797	166.48
Gompertz model	3.906	21.269	0.029	1.397	0.986	4.799	144.42
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	9.544	119.13	0.037	1.366	0.971	13.901	223.89
GO model	6.403	52.397	0.025	1.072	0.987	8.471	186.11
DSS model	12.008	203.90	0.049	1.419	0.950	14.437	248.61
ISS model	6.318	57.407	0.026	0.982	0.986	7.661	190.31
Logistic model	10.047	161.97	0.044	3.783	0.960	19.702	238.02
Gompertz model	7.556	80.811	0.031	2.237	0.980	10.744	206.04

Table 9
Parameter estimations of models for corrected faults (DS2).

LSE							
Models (High-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.026	–	0.418	–	–	–
GO model	4.71×10^3	5.79×10^{-4}	–	–	–	–	–
DSS model	160.07	0.064	–	–	–	–	–
ISS model	140.60	0.084	–	–	5.034	–	–
Logistic model	131.68	0.121	–	15.26	–	–	–
Gompertz model	149.91	0.936	–	0.016	–	–	–
Proposed PPQ model #1	5.42×10^3	5.17×10^{-4}	40	–	–	0.087	3.15
Proposed PPQ model #2	143.51	0.081	40	–	5.334	0.063	2.32
Models (Low-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	0.857	–	0.085	–	–	–
GO model	453.94	0.014	–	–	–	–	–
DSS model	250.30	0.075	–	–	–	–	–
ISS model	369.72	0.028	–	–	0.741	–	–
Logistic model	251.40	0.088	–	8.277	–	–	–
Gompertz model	249.49	0.934	–	0.052	–	–	–
Proposed PPQ model #1	473.10	0.013	40	–	–	0.167	6.23
Proposed PPQ model #2	335.24	0.030	40	–	0.673	0.119	4.61
MLE							
Models (High-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.061	–	0.489	–	–	–
GO model	931.44	3.01×10^{-3}	–	–	–	–	–
DSS model	159.07	0.063	–	–	–	–	–
ISS model	139.72	0.079	–	–	6.193	–	–
Logistic model	131.34	0.124	–	21.39	–	–	–
Gompertz model	148.36	0.940	–	0.028	–	–	–
Proposed PPQ model #1	1.06×10^3	2.70×10^{-3}	40	–	–	0.079	2.90

(*continued on next page*)

Table 9 (*continued*).

Proposed PPQ model #2	155.17	0.067	40	–	4.138	0.082	3.08
Models (Low-priority)	a	b	c	–	ψ	μ	λ
Duane model	–	0.983	–	0.187	–	–	–
GO model	643.42	9.45×10^{-3}	–	–	–	–	–
DSS model	276.42	0.067	–	–	–	–	–
ISS model	595.90	0.011	–	–	0.007	–	–
Logistic model	273.54	0.082	–	8.691	–	–	–
Gompertz model	242.61	0.931	–	0.052	–	–	–
Proposed PPQ model #1	894.64	6.27×10^{-3}	40	–	–	0.144	5.13
Proposed PPQ model #2	879.45	6.41×10^{-3}	40	–	0.135	0.136	4.96

Table 10
Performance comparisons of models for corrected faults (DS2).

LSE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	4.549	29.602	0.037	1.109	0.980	6.373	–
GO model	4.590	30.796	0.036	1.480	0.979	7.821	–
DSS model	4.698	29.840	0.036	0.650	0.980	6.796	–
ISS model	4.912	32.464	0.037	0.798	0.978	9.016	–
Logistic model	5.200	35.149	0.038	2.250	0.976	9.102	–
Gompertz model	5.110	37.686	0.042	0.648	0.974	10.215	–
Proposed model #1	4.162	23.464	0.032	1.376	0.984	5.068	–
Proposed model #2	3.658	18.157	0.028	0.667	0.988	5.340	–
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	6.307	62.501	0.029	3.806	0.958	7.933	–
GO model	7.404	74.645	0.031	3.391	0.982	12.322	–
DSS model	7.471	92.240	0.035	2.906	0.978	9.892	–
ISS model	7.176	84.454	0.034	2.767	0.980	12.262	–
Logistic model	8.880	145.21	0.043	6.570	0.965	12.329	–
Gompertz model	7.106	75.067	0.031	3.773	0.982	8.601	–
Proposed model #1	5.940	51.761	0.026	3.169	0.988	7.218	–
Proposed model #2	5.984	51.364	0.026	3.033	0.988	7.159	–
MLE							
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	4.936	36.993	0.042	0.898	0.975	8.226	170.09
GO model	5.187	34.048	0.039	1.368	0.974	6.571	166.28
DSS model	4.912	31.560	0.036	0.754	0.982	5.975	161.09
ISS model	4.789	35.123	0.041	0.481	0.976	9.714	167.71
Logistic model	4.754	33.299	0.039	1.358	0.977	6.654	165.25
Gompertz model	5.458	43.315	0.045	1.692	0.971	11.050	177.35
Proposed model #1	4.321	25.664	0.034	1.427	0.983	5.122	150.27
Proposed model #2	2.944	14.568	0.025	0.767	0.990	3.863	127.22
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	7.965	111.277	0.039	3.493	0.973	12.994	218.71
GO model	8.505	94.247	0.034	2.936	0.977	13.389	211.07
DSS model	11.22	178.62	0.047	4.646	0.957	13.216	240.48
ISS model	8.331	90.751	0.034	3.182	0.978	13.332	209.33
Logistic model	10.90	191.23	0.049	4.913	0.954	15.357	243.62
Gompertz model	10.88	178.37	0.047	4.099	0.957	21.371	240.42
Proposed model #1	6.632	68.593	0.030	2.797	0.984	8.190	196.43
Proposed model #2	6.402	67.981	0.030	2.766	0.983	8.321	196.04

Finally, Fig. 10 shows the Pred(l) plots of the selected models for different l levels ranging from 2^{-8} to 1. In Fig. 10), the Pred(0.25) values for the Duane model, the GO model, the DSS model, the ISS model, the Logistic model, the Gompertz model, and the proposed PPQ models #1 and #2 are 0.85 0.75, 0.78, 0.89, 0.78, 0.92, 0.82, and 0.92, respectively. Fig. 10(d) shows that 92% of estimates from the ISS model and PPQ model #2 are within 25% of the actual data. Overall, the Pred(l) plots show that PPQ model #2 performs well at different l levels. Our experimental results show that PPQ model #2 provide a better fit with DS3 and predict future FCPs well.

4.6. Threats to validity

In this subsection, we describe the threats to internal, external, construct, and conclusion validity for our experiment. These four kinds of threats to validity in this study will be discussed as follows:

Internal validity concerns whether uncontrolled factors exist that we did not take into account. The failure data collection is one threat to our internal validity. There are few datasets that have recorded fault detection, fault correction and fault priority level information. However, only Bugzilla can provide complete

Table 11
Parameter estimation results of models for detected faults (DS3).

Parameter estimation	LSE			
Models (High-priority)	*a*	*b*	*ψ*	*k*
Duane model	–	0.960	–	0.797
GO model	158.50	0.008	–	——
DSS model	35.67	0.118	–	——
ISS model	40.79	0.083	2.282	——
Logistic model	32.17	0.181	–	11.615
Gompertz model	36.01	0.900	–	0.038
Models (Low-priority)	*a*	*b*	*ψ*	*k*
Duane model	–	1.057	–	0.473
GO model	2215.3	0.001	–	——
DSS model	94.80	0.097	–	——
ISS model	100.56	0.093	3.729	—
Logistic model	78.95	0.171	–	13.423
Gompertz model	97.81	0.918	–	0.036
Parameter Estimation	MLE			
Models (High-priority)	*a*	*b*	*ψ*	*k*
Duane model	–	1.034	–	1.043
GO model	106.85	0.012	–	——
DSS model	36.68	0.111	–	——
ISS model	37.85	0.107	3.347	——
Logistic model	32.96	0.189	–	17.379
Gompertz model	33.35	0.881	–	0.018
Models (Low-priority)	*a*	*b*	*ψ*	*k*
Duane model	–	1.134	–	0.633
GO model	291.06	0.010	–	——
DSS model	104.20	0.087	–	——
ISS model	95.87	0.108	5.260	——
Logistic model	82.34	0.174	–	16.492
Gompertz model	93.20	0.911	–	0.037

Table 12
Performance comparisons of models for detected faults (DS3).

Parameter Estimation	LSE						
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	0.871	1.203	0.030	0.867	0.986	1.208	–
GO model	0.789	0.907	0.025	0.896	0.989	1.018	–
DSS model	0.965	1.412	0.031	0.972	0.983	1.342	–
ISS model	0.737	0.787	0.024	0.742	0.991	0.923	–
Logistic model	1.176	2.282	0.040	1.062	0.973	1.933	–
Gompertz model	0.983	1.455	0.032	0.969	0.983	1.563	–
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	1.567	3.943	0.023	0.896	0.992	2.023	–
GO model	1.741	4.835	0.025	0.802	0.990	2.328	–
DSS model	2.270	6.815	0.030	1.432	0.986	2.934	–
ISS model	1.418	3.152	0.020	0.812	0.993	1.814	–
Logistic model	1.731	3.882	0.023	1.388	0.992	2.015	–
Gompertz model	1.552	3.219	0.021	0.855	0.993	1.828	–
Parameter Estimation	MLE						
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	1.357	2.555	0.044	0.723	0.970	2.493	30.26
GO model	0.798	1.026	0.027	0.789	0.988	1.031	12.72
DSS model	0.916	1.242	0.029	0.675	0.985	1.134	15.06
ISS model	0.765	0.846	0.025	0.642	0.990	0.938	11.68
Logistic model	1.468	3.239	0.048	0.734	0.962	2.037	36.91
Gompertz model	1.118	1.746	0.035	0.754	0.980	1.334	19.11
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	AIC
Duane model	1.984	7.948	0.033	1.577	0.984	4.415	62.04
GO model	2.335	8.563	0.033	1.179	0.982	4.091	64.13
DSS model	2.200	7.574	0.038	1.567	0.984	3.262	60.69
ISS model	1.597	4.684	0.025	0.866	0.990	3.068	47.24
Logistic model	2.004	5.815	0.028	1.099	0.988	2.542	53.29
Gompertz model	1.898	5.015	0.025	0.882	0.990	3.117	49.15

Table 13
Parameter estimations of models for corrected faults (DS3)

Parameter Estimation	LSE						
Models (High-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.236	–	1.858	–	–	–
GO model	543.14	1.93×10^{-3}	–	–	–	–	–
DSS model	50.54	0.069	–	–	–	–	–
ISS model	37.84	0.119	–	–	7.956	–	–
Logistic model	38.84	0.132	–	13.06	–	–	–
Gompertz model	35.01	0.899	–	0.010	–	–	–
Proposed PPQ model #1	747.07	1.36×10^{-3}	7	–	–	0.211	1.313
Proposed PPQ model #2	32.74	0.133	7	–	5.871	0.158	1.029
Models (Low-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.029	–	0.428	–	–	–
GO model	897.22	2.97×10^{-3}	–	–	–	–	–
DSS model	105.20	0.086	–	–	–	–	–
ISS model	114.73	0.096	–	–	7.183	–	–
Logistic model	79.43	0.190	–	20.93	–	–	–
Gompertz model	79.21	0.885	–	0.016	–	–	–
Proposed PPQ model #1	673.45	3.92×10^{-3}	7	–	–	0.393	2.531
Proposed PPQ model #2	99.49	0.097	7	–	4.737	0.413	2.715
Parameter Estimation	MLE						
Models (High-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.238	–	1.856	–	–	–
GO model	7851.8	1.31×10^{-4}	–	–	–	–	–
DSS model	47.88	0.070	–	–	–	–	–
ISS model	37.72	0.125	–	–	9.513	–	–
Logistic model	32.04	0.180	–	20.18	–	–	–
Gompertz model	31.81	0.869	–	0.002	–	–	–
Proposed PPQ model #1	10011.4	1.03×10^{-4}	7	–	–	0.196	1.277
Proposed PPQ model #2	33.14	0.141	7	–	8.281	0.187	1.193
Models (Low-priority)	a	b	c	k	ψ	μ	λ
Duane model	–	1.029	–	0.429	–	–	–
GO model	1540.34	1.70×10^{-3}	–	–	–	–	–
DSS model	104.741	0.086	–	–	–	–	–
ISS model	118.115	0.093	–	–	6.920	–	–
Logistic model	81.5147	0.176	–	21.64	–	–	–
Gompertz model	82.2095	0.889	–	0.017	–	–	–
Proposed PPQ model #1	2571.36	1.01×10^{-3}	7	–	–	0.387	2.498
Proposed PPQ model #2	11.0115	0.955	7	–	5.720	0.426	2.681

information about faults, such as detection time, correction time, and priority level, which is an open bug-tracking system and is widely used in software engineering research. Another threat to internal validity is the selection of subjects. However, fault reports and the development progress of OSS are usually accomplished by many volunteer participators (Samoladas et al., 2010). The fault reports in the system are usually duplicate, invalid or unreasonable. Hence, we carefully eliminated these data from our datasets to improve the correctness. Note that we can obtain all the required information for experiments from the failure data of CSS. In addition, the selected SRGMs for comparison are another threat to our internal validity.

External validity concerns whether the same experiment result can be generalizable to other settings or situations. To reduce the threats to external validity for our experiments, we apply three real failure data from the Mozilla Core and Eclipse project (OSS) and a commercial software of embedded system (CSS) to our proposed model because there is significant difference in the FCP between OSS and CSS. Not only are their architectures distinct, but the composition of the developers are also different. Therefore, the threats to external validity should be minimal, and the results of our experiments can be generalized to other software projects.

Construct validity concerns whether the measure can exactly reflect or access the underlying construct that is intended to be measured. To examine whether our proposed model can provide a more accurate estimation of the fault correction process, we use some criteria to compare the performance of the traditional SRGMs. These criteria are useful for diminishing the threat to construct validity, it is noted that most of the selected criteria are extensively adopted for evaluation in many software reliability studies (Huang and Huang, 2008).

Conclusion validity concerns whether the conclusions we reach about relationships in our experimental results are reasonable. The random heterogeneity of subjects is one threat to conclusion validity. To ensure that the selected systems were non-homogeneous, we collected the actual failure data from three different software projects in this study. Two of them are large-scale OSS projects with highly experienced core contributors, and the other one is a medium-scale CSS project developed by developers in a commercial company. Three real failure datasets are non-homogeneous.

5. Conclusion

With the rapid development of new technology, people are increasingly dependent on the services provided by software in their daily lives. Thus, the reliability estimation for releasing mission-critical and safety-critical application systems is vital. Previous research has primarily used SRGMs to evaluate and predict software reliability, and the detected faults are intuitively assumed to be immediately removed. However, developers must spend time analyzing the root causes of faults in the real world. Therefore, to consider the time lag from debugging works, several researchers have integrated queueing theory into SRGMs. As a result, in these models, the detected faults are removed

Table 14
Performance comparisons of models for corrected faults (DS3).

Parameter Estimation	LSE						
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	1.470	3.055	0.050	1.636	0.961	3.004	——
GO model	1.383	2.666	0.047	2.183	0.966	2.948	——
DSS model	1.078	1.712	0.039	0.549	0.978	1.355	——
ISS model	1.190	1.980	0.041	0.714	0.975	1.731	——
Logistic model	1.500	3.099	0.053	2.082	0.960	1.891	——
Gompertz model	1.356	3.001	0.050	0.523	0.962	2.572	——
Proposed model #1	0.981	1.431	0.036	1.833	0.982	1.144	——
Proposed model #2	0.742	0.794	0.027	0.835	0.990	0.940	——
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	2.438	8.340	0.033	1.203	0.984	4.995	——
GO model	2.409	9.392	0.035	1.281	0.981	4.985	——
DSS model	2.435	7.820	0.033	1.178	0.985	3.235	——
ISS model	2.346	9.028	0.036	1.284	0.982	4.778	——
Logistic model	2.284	8.258	0.033	1.149	0.984	3.048	——
Gompertz model	2.901	12.252	0.040	1.701	0.976	4.824	——
Proposed model #1	2.192	6.934	0.032	1.058	0.985	3.938	——
Proposed model #2	1.111	1.920	0.016	0.617	0.996	1.360	——
Parameter Estimation	MLE						
Models (High-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	1.210	2.326	0.046	1.179	0.970	2.115	27.635
GO model	1.054	1.535	0.037	1.629	0.980	1.786	15.999
DSS model	1.105	2.048	0.044	0.973	0.974	2.195	24.073
ISS model	1.322	2.968	0.053	1.043	0.962	2.441	34.462
Logistic model	1.354	2.689	0.049	1.319	0.966	1.737	31.697
Gompertz model	1.475	2.926	0.050	1.348	0.963	1.879	34.060
Proposed model #1	0.966	1.396	0.035	0.883	0.982	1.331	13.335
Proposed model #2	0.873	1.167	0.032	0.804	0.985	1.121	8.318
Models (Low-priority)	MAE	MSE	TS-U1	TS-U2	R^2	Var.	*AIC*
Duane model	2.358	7.934	0.032	1.035	0.984	4.827	59.917
GO model	2.603	10.05	0.036	1.527	0.980	5.402	67.556
DSS model	2.467	8.423	0.033	1.225	0.985	4.099	58.053
ISS model	2.125	7.356	0.032	0.932	0.985	4.017	57.798
Logistic model	2.688	10.90	0.039	1.590	0.978	4.578	68.828
Gompertz model	3.049	13.83	0.042	1.532	0.973	5.482	85.476
Proposed model #1	2.006	6.816	0.030	0.848	0.987	3.781	55.662
Proposed model #2	1.454	3.133	0.020	0.707	0.993	1.737	33.905

by following the FCFS rule. However, this assumption may not be appropriate for the real world because debuggers may make different decisions for faults with various properties. Practically, each detected fault will be assigned a priority level and faults with higher priority are corrected sooner than faults with lower priority. Our proposed models and related preemptive policies allow high-priority faults to immediately interrupt the process of dealing with low-priority faults even if they are currently processed and return them to the head of the waiting queue. Experimental results show that the proposed PPQ model provides a better fit to the observed data than traditional SRGMs and predicts future behavior well. Note that our proposed PPQ model can also calculate and provide effective information, such as the average waiting time, the average response time, and the utilization of debuggers in the removal process, for the project manager's reference.

Our future works can be divided into selected topics. First, we plan to continue extending our methodology to more than two priority classes and/or discovering more classical SRGMs with different prioritization policies (Wang et al., 2015). We will also investigate the case where debuggers do not have equal capability to correct the detected faults. Additionally, imperfect debugging (or imperfect repair) phenomena usually occur in the real world (Gokhale, 2007; Yao and Zhang, 2018). Thus this issue should also be taken into consideration. On the other hand, we plan to collect larger and more diverse datasets. In this paper, the three OSS datasets we used are collected, but we plan to have more open- and/or closed-source software datasets to evaluate the performance of our proposed models. Lastly, we plan to investigate the introduction of the express queueing model. The detected faults of the software can be delivered to the express queue or allocated to the express queue debuggers if the required service time is below a certain value. We plan to study the above mentioned issues, and publish the results in the near future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The work described in this paper was supported by the Ministry of Science and Technology, Taiwan, under Grants MOST 108-2221-E-007-033-MY3 and MOST 110-2221-E-007-035-MY3.

Appendix A. DS1

See Table A.15.

Appendix B. DS2

See Table B.16.

Table A.15
DS1.

Time (Weeks)	# of high priority faults detected	# of high priority faults removed	# of low priority faults detected	# of low priority faults removed
1	4	1	1	1
2	7	5	1	1
3	9	7	2	1
4	14	9	4	3
5	14	11	4	3
6	15	12	4	3
7	18	14	5	4
8	20	17	7	4
9	21	18	8	5
10	22	20	8	6
11	22	20	8	6
12	25	22	8	6
13	25	23	8	6
14	26	23	9	6
15	26	24	10	6
16	28	25	11	7
17	28	26	12	8
18	30	27	13	11
19	30	28	13	11
20	31	29	13	11
21	32	29	14	11
22	33	30	14	11
23	34	30	14	11
24	34	31	14	11
25	36	32	16	13
26	36	34	16	13
27	37	34	17	13
28	38	34	19	14
29	39	36	19	16
30	40	36	19	17
31	40	37	19	17
32	40	38	19	17
33	41	38	19	17
34	41	38	20	18
35	41	38	20	18
36	41	39	20	18
37	43	40	20	18
38	44	41	20	18
39	46	43	21	19
40	48	47	21	19
41	48	47	21	19
42	52	49	21	20
43	53	49	21	20
44	54	51	22	20
45	56	51	25	20

Table B.16
DS2.

Time (Weeks)	# of high priority faults detected	# of high priority faults removed	# of low priority faults detected	# of low priority faults removed
1	0	0	3	0
2	4	1	5	1
3	5	4	9	4
4	8	8	35	17
5	14	9	46	33
6	26	17	51	45
7	30	24	53	47
8	30	24	53	48
9	30	25	54	49
10	33	25	55	50
11	36	26	56	51
12	39	28	73	65
13	40	30	80	66
14	40	33	80	74
15	40	33	87	80
16	43	35	90	84

(*continued on next page*)

Table B.16 (*continued*).

Time (Weeks)	# of high priority faults detected	# of high priority faults removed	# of low priority faults detected	# of low priority faults removed
17	47	40	97	91
18	53	42	115	104
19	53	46	125	117
20	58	53	126	119
21	62	57	130	122
22	73	67	137	130
23	78	72	143	136
24	80	76	143	139
25	81	78	145	141
26	81	78	146	142
27	81	78	146	142
28	93	83	155	148
29	94	86	155	149
30	95	87	160	151
31	97	89	160	156
32	101	92	168	159
33	101	92	168	161
34	103	92	170	162
35	105	100	178	166
36	108	103	189	170
37	109	105	191	175
38	109	107	194	181
39	112	111	201	189
40	113	112	204	194
41	113	112	207	200
42	114	113	210	200
43	120	118	212	209
44	121	119	216	209
45	122	119	218	211
46	126	124	231	224

Table C.17
DS3.

Time (Week)	# of high-priority faults detected	# of high-priority faults removed	# of low-priority faults detected	# of low-priority faults removed
1	0	0	3	1
2	1	0	4	2
3	1	1	6	5
4	4	2	9	8
5	6	3	13	10
6	6	4	17	14
7	8	5	18	16
8	10	7	20	19
9	10	8	22	20
10	13	9	23	21
11	14	11	25	23
12	14	13	28	26
13	16	14	31	29
14	16	15	35	33
15	18	16	37	34
16	18	17	41	39
17	19	17	46	44
18	20	17	51	46
19	23	20	55	51
20	23	21	56	52
21	24	22	56	53
22	24	22	59	56
23	25	23	60	58
24	28	23	62	60
25	29	25	65	63
26	29	26	69	67
27	29	28	71	69
28	30	28	73	71

Appendix C. DS3

See Table C.17.

References

Akaike, H., 1974. A new look at the statistical mode identification. IEEE Trans. Automat. Control 19 (6), 716–723.

Antoniol, G., Cimitile, A., Di Lucca, G.A., Di Penta, M., 2004. Assessing staffing needs for a software maintenance project through queuing simulation. IEEE Trans. Softw. Eng. 30 (1), 43–58.

Balsamo, S., Personè, N., Inverardi, P., 2003. A review on queueing network models with finite capacity queues for software architectures performance prediction. Perform. Eval. 51, 269–288.

Bondi, A., Buzen, J., 1984. The response times of priority classes under preemptive resume in M/G/m queues. ACM SIGMETRICS Perform. Eval. Rev. 12 (3), 195–201, New York, NY, USA.

Britton, T., Jeng, L., Carver, G., Cheak, P., Katzenellenbogen, T., 2013. Reversible debugging software. In: Technical Report. University of Cambridge-Judge Business School, Cambridge, U.K.

Bugzilla, Available at: https://bugzilla.mozilla.org/. (Accessed 30 April 2016).

Chai, T., Draxler, R., 2014. Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. Geosci. Model Dev. 7 (3), 1247–1250.

Conte, S.D., Dunsmore, H.E., Shen, V.Y., 1986. Software Engineering Metrics and Models. Benjamin Cummings, Redwood City, CA, USA.

Dohi, T., Matsuoka, T., Osaki, S., 2002. An infinite server queuing model for assessment of the software reliability. Electron. Commun. Japan, Part 3 85 (3), 43–51.

Dohi, T., Osaki, S., Trivediy, K., 2004. An infinite server queueing approach for describing software reliability growth - Unified modeling and estimation framework. In: Proceeding of the 11th Asia-Pacific Software Engineering Conference (APSEC), Busan, South Korea, pp. 110–119.

Eclipse Bugzilla, Available at: https://bugs.eclipse.org/bugs/ (Accessed 30 April 2016).

Fenton, N.E., Pfleeger, S.L., 1998. Software Metrics: A Rigorous and Practical Approach, second ed. PWS Pub, Boston, MA, USA.

Gao, R., Wong, W.E., 2019. MSeer - An Advanced technique for locating multiple bugs in parallel. IEEE Trans. Softw. Eng. (TSE) 45 (3), 301–318.

Gokhale, S.S., 2007. Architecture-based software reliability analysis: Overview and limitations. IEEE Trans. Dependable Secur. Comput. 4 (1), 32–40.

Gokhale, S.S., Mullen, R.E., 2006. Queuing models for field defect resolution process. In: Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE), Raleigh, NC, pp. 353–362.

Ho, S., Xie, M., 1998. The use of ARIMA models for reliability forecasting and analysis. Comput. Ind. Eng. 35, 213–216.

Holden, K., Peel, D.A., Thompson, J.L., 1991. Economic Forecasting: An Introduction. Cambridge Univ. Press, Cambridge, U.K.

Hsu, C.J., Huang, C.Y., 2014. Optimal weighted combinational models for software reliability estimation and analysis. IEEE Trans. Reliab. 63 (3), 731–749.

Hu, Q.P., Xie, M., Ng, S.H., 2006a. Software reliability prediction improvement with prior information incorporated. In: Proceedings of the 12th ISSAT International Conference on Reliability and Quality in Design, Chicago, USA, pp. 303–307.

Hu, Q., Xie, M., Ng, S., 2006b. Early software reliability prediction with ANN models. In: Proceedings of the 12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), Riverside, CA, USA, pp. 303–307.

Huang, C.Y., Huang, W.C., 2008. Software reliability analysis and measurement using finite and infinite server queueing models. IEEE Trans. Reliab. 57 (1), 192–203.

Huang, C.Y., Hung, T.Y., Hsu, C.J., 2009. Software reliability prediction and analysis using queueing models with multiple change-points. In: Proceeding of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI), Shanghai, China, pp. 212–221.

Huang, C.Y., Kuo, T.Y., 2017. Queueing-theory-based models for software reliability analysis and management. IEEE Trans. Emerg. Top. Comput. 5 (4), 540–550.

Huang, C.Y., Lin, C.T., 2006. Software reliability analysis by considering fault dependency and debugging time lag. IEEE Trans. Reliab. 55 (3), 436–450.

Inoue, S., Yamada, S., 2003. A software reliability growth modeling based on infinite server queueing theory. In: Proceeding of the 9th ISSAT International Conference on Reliability and Quality in Design (QRD), Waikiki, HI, USA, pp. 305–309.

Inoue, S., Yamada, S., 2006. An extended delayed S-shaped software reliability growth model based on infinite server queuing theory. Reliab. Model. Anal. Optim. 357–372.

Kanoun, K., Laprie, J., 1994. Software reliability trend analyses from theoretical to practical considerations. IEEE Trans. Softw. Eng. 20 (9), 740–747.

Kanoun, K., Martini, Souza, J. de, 1991. A method for software reliability analysis and prediction application to the TROPICO-R switching system. IEEE Trans. Softw. Eng. 17 (4), 334–344.

Kapur, P.K., Aggarwal, A.G., Kumar, R., 2010a. A unified approach for discrete software reliability growth model for faults of different severity using infinite server queuing model. Commun. Dependability Qual. Manag. Int. J. 13 (4), 66–81.

Kapur, P.K., Anand, S., Inoue, S., Yamada, S., 2010b. A unified approach for developing software reliability growth model using infinite server queuing model. Int. J. Reliab. Qual. Safety Eng. 17 (5), 401–424.

Keller, G., Warrack, B., 1999. Statistics for Management and Economics. Duxbury.

Kleinrock, L., 2016. Queueing Systems. John Wiley & Sons.

Kong, X., Zhang, L., Wong, W.E., Li, B., 2018. The impacts of techniques, programs and tests on automated program repair: An empirical study. J. Syst. Softw. 137, 480–496.

Laplante, P.A., Ahmad, N., 2009. Pavlov's Bugs: MAtching repair policies with rewards. IT Prof. 11 (4), 45–51.

Li, P.L., Herbsleb, J., Shaw, M., 2005. Forecasting field defect rates using a combined time-based and metrics-based approach: A case study of OpenBSD. In: Proceeding of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE), Chicago, IL, USA, pp. 193–202.

Li, Y., Wong, W.E., Lee, S.Y., Wotawa, F., 2019. Using tri-relation networks for effective software fault-proneness prediction. IEEE Access 7, 63066–63080.

Lim, D.E., Kim, T.S., 2014. Modeling discovery and removal of security vulnerabilities in software system using priority queueing models. J. Comput. Virol. Hacking Tech. 10 (2), 109–114.

Lin, J.S., 2017. Software Reliability Analysis using Preemptive Priority Queueing Models and Rate-Based Simulation Procedures. (M.S. Thesis). Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan.

Lyu, M.R., 1996. HandBook of Software Reliability Engineering. McGraw-Hill, Inc., Hightstown, NJ.

Makaveli, D., 2017. Open source software versus commercial software: An in-depth analysis of the issues. Available at: http://www.academia.edu/17783834/open_source_software_versus_commercial_software_an_in-depth_analysis_of_the_issues. (Accessed 18 July 2017).

Mockus, A., Fielding, R., Herbsleb, J., 2002. Two case studies of open source software development: Apache and mozilla. ACM Trans. Softw. Eng. Methodol. 11 (3), 309–346.

Musa, J.D., Iannino, A., Okumoto, K., 1987. Software Reliability, Measurement, Prediction, and Application. McGraw-Hill.

Pham, H., 2006. System Software Reliability, Reliability Engineering Series. Springer, London, UK.

Pillai, K., Nair, V.S., 1997. A model for software development effort and cost estimation. IEEE Trans. Softw. Eng. 23 (8), 485–497.

Samoladas, I., Angelis, L., Stamelos, I., 2010. Survival analysis on the duration of open source projects. Inf. Softw. Technol. 52 (9), 902–922.

Schneidewind, N., 2010. Software development queuing model. J. Aerosp. Comput. Inf. Commun. 7, 308–321.

Shin, M., Goel, A.L., 2000. Empirical data modeling in software engineering using radial basis functions. IEEE Trans. Softw. Eng. 26 (6), 567–576.

Shortle, J.F., Thompson, J.M., Gross, D., Harris, C.M., 2018. Fundamentals of Queueing Theory, 5th Edition John Wiley and Sons.

Sukhwani, H., Alonso, J., Trivedi, K.S., Mcginnis, I., 2016. Software reliability analysis of NASA space flight software: A practical experience. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS), Vienna, Austria, pp. 386–397.

Tausworthe, R.C., Lyu, M.R., 1996. A generalized technique for simulation software reliability. IEEE Softw. 13 (2), 77–88.

Tijms, H., 2003. A First Course in Stochastic Models, 1st Edition John Wiley & Sons Ltd.

Vaidyanathan, K., Trivedi, K.S., 2005. A comprehensive model for software rejuvenation. IEEE Trans. Dependable Secur. Comput. 2 (Np. 2), 124–137.

Wang, J., Baron, O., Scheller-Wolf, A., 2015. M/M/c Queue with two priority classes. Oper. Res. 63 (3), 733–749.

Wong, W.E., Debroy, V., Surampudi, A., Kim, H., Siok, M.F., 2010. Recent catastrophic accidents: Investigating how software was responsible. In: Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, pp. 14–22.

Xie, M., 1991. Software Reliability Modelling, first ed. World Scientific, Singapore.

Xie, M., Hong, G., Wohlin, C., 1999. Software reliability prediction incorporating information from a similar project. J. Syst. Softw. 49 (1), 43–48.

Yang, K.Z., 1996. An Infinite Server Queueing Model for Software Readiness Assessment and Related Performance Measures. Ph. D. Dissertation. Dept. Elect. Eng. Comput. Sci. Syracuse Univ. Syracuse, NY, USA.

Yao, D., Zhang, N., 2018. A queue theory-based approach to software reliability assessment. In: 2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), pp. 244–250.

Ye, Y., Kishida, K., 2003. Toward an understanding of the motivation open source software developers. In: Proceedings of the 25th IEEE International Conference on Software Engineering (ICSE), Oregon, Portland, pp. 419–429 May.

Zhang, N., 2015. Queue-based FDP and FCP analysis with detection effort and correction effort. J. Inf. Comput. Sci. 12 (1), 21–29.

Zhang, N., Cui, G., Liu, H.W., 2011. A finite queuing model with generalized modified Weibull testing effort for software reliability. In: Proceedings of 2011 International Conference on Computer Science and Network Technology (ICCSNT 2011), pp. 401–406 Dec. 2011.