IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 26 June 2014; revised 5 December 2014; accepted 28 December 2014. Date of current version 6 December 2017.

Digital Object Identifier 10.1109/TETC.2014.2388454

Queueing-Theory-Based Models for Software Reliability Analysis and Management

CHIN-YU HUANG[1], (Member, IEEE), AND TZU-YU KUO[2]

[1]Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan
[2]Department of Engineering, Garmin Corporation, New Taipei City, Taiwan

CORRESPONDING AUTHOR: C.-Y. HUANG (cyhuang@cs.nthu.edu.tw)

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 103-2220-E-007-022.

ABSTRACT Software reliability is one of the most important internal attributes of software systems. Over the past three decades, many software reliability growth models have been proposed and discussed. Some research has also shown that the fault detection and removal processes of software can be described and modeled using an infinite-server queueing system. But, there is practically no company that can afford unlimited resources to test and correct detected faults in the real world. Consequently, the number of debuggers should be limited, not infinite. In this paper, we propose an extended finite-server-queueing (EFSQ) model to analyze the fault removal process of the software system. Numerical examples based on real project data are illustrated. Evaluation results show that our proposed EFSQ model has a fairly accurate prediction capability of software reliability and also depict the real-life situation of software development activities more faithfully. Finally, the applications of our proposed model to project management are also presented. Our proposed model can provide a theoretically effective technique for managing the necessary activities of testing and debugging in software project management.

INDEX TERMS Software testing and debugging, queueing theory, software reliability growth model (SRGM), project management, non-homogeneous Poisson process (NHPP).

I. INTRODUCTION

Presently, software plays an important role in the life of people, and people need reliable software. But developing reliable software is not an easy thing since the software development process has very high variance. There are a lot of techniques to improve the reliability and availability of software systems, such as the software rejuvenation, update release, etc [1]. It is worth noting that rejuvenation works when the software reliability decreases over time. Additionally, software aging phenomenon should be non-negligible Software aging is usually a consequence of software faults. Consequently, the effects of software aging can be reduced or improved if all detected faults are corrected perfectly before releasing to the users.

Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment, and it is one of the most important internal attributes of software systems [2]. Over the past three decades, many *Software Reliability Growth Models* (SRGMs) or some simulation approaches were proposed for software reliability prediction and estimation [3], [4]. But most of the previous research assumed that detected faults can be immediately fixed. In actuality, this assumption may not be realistic, since the time to remove detected fault(s) typically depends on the complety of the detected fault(s), the knowledge, skills and practical experiences of the personnel, the size of the development team, the testing/debugging technique(s) being used, and so on [5].

In the past, some research has also shown that queueing theory (or the queueing model) can be used to describe various activities in the software development life cycle (SDLC) [6]–[20]. For instance, Balsamo et al. [7] discussed and presented the application of queueing network models with finite capacity queues and blocking as the performance models of software architectures. Antoniol et al. [12] once proposed a queueing theory-based approach to plan and control project staffing in a distributed multiphase maintenance process. Dohi et al. [19] also presented an interpretation of traditional SRGMs by treating the software failure occurrence process as an *Infinite-Server-Queueing* (ISQ).

It can be found that most of the previous research assumed that the queueing model for describing software development

activities is an *Infinite Server Queueing Model*. But there is practically no company that can afford unlimited resources to test and correct software faults in the real world. Consequently, the assumption of the ISQ model has to be properly modified. Based on our past studies [10], in this paper we further propose an *Extended Finite-Server-Queueing* (EFSQ) model to analyze the fault removal process of the software system, and to estimate the software reliability. Experiments will be performed based on two real project data, and experimental results will be analyzed and discussed in detail. The applications of the proposed model to project management are also presented.

The remainder of this paper is organized as follows. Section II gives a survey of describing software reliability growth using ISQ models in the literature. In Section III, we study and show how to derive an EFSQ model in exceptional detail. In Section IV, based on real software failure data, experiments will be performed, performance evaluations among some selected SRGMs are presented, and threats to validity are also discussed. We further illustrate and show in Section V how the prediction and analysis results can provide useful information that can be instrumental to various management decisions and testing-resource allocations. Finally, some conclusions are given in Section VI.

II. BACKGROUND AND RELATED WORKS

During the SDLC process, testing and debugging are different processes. Testing is the process of finding faults in software, but debugging is the process of fixing detected faults. Practically developers need some time to accurately determine the root causes of reported faults. Thus, the time delayed by the fault detection and correction processes should not be negligible [5]. In the past, some research has shown that classical SRGMs or ISQ models can be used to describe the activities of fault detection and removal in the SDLC. For example, Yamada et al. [4] proposed a delayed S-shaped model, and the model was designed to capture the fault removal phenomenon of software systems. Gokhale and Mullen [8] also considered the effect of queueing system structures, priority levels and priority disciplines in differential mean times to resolve defects of different severities. Additionally, Kapur et al. [15] proposed a unified modeling approach of applying the ISQ theory and defined fault removal lags as the random variables.

It is worth noting that Inoue and Yamada [11] proposed an ISQ model, considering the time distribution of the fault-isolation process, based on a concept of classical delayed S-shaped SRGM. They proposed and summarized the relationships between the ISQ model and the existing NHPP models as follows:

$$P\{N(t)=n\} = \frac{\left[\int_0^t F(t-x)d\Lambda(x)\right]^n}{n!} \times \exp\left[-\int_0^t F(t-x)d\Lambda(x)\right] \quad (1)$$

and

$$M(t) = \int_0^t F(t-x)d\Lambda(x) \quad (2)$$

where $N(t)$ denotes a counting process representing the cumulative number of faults detected by the time t, $M(t)$ is the expected cumulative number of faults isolated (or detected) up to time t and is non-decreasing with respect to time t, $F(t)$ is the time distribution function of the fault-isolation process and $\Lambda(t)$ is the expected cumulative number of failures observed up to time t.

Huang et al. [10] also proposed and used the ISQ model to predict software reliability under perfect and imperfect debugging environments. They once defined the probability that debuggers or developers will be available at time and took it into the consideration when deriving the mathematical model of FSQ. They assumed that the probability is $P(c) = \sum_{h=0}^{c-1} (\vartheta^h e^{-\vartheta}/h!)$ and $(\vartheta^h e^{-\vartheta})/h!$ is a Poisson distribution. Note that c is the number of debuggers, h is number of faults in the correction, and ϑ is a positive constant. But all debuggers have practically identical capabilities, and a debugger could be chosen at random if multiple debuggers are available.

From above discussions, it can be found that most of the published studies which usually assumed the queueing model for describing fault detection and removal activities, is an ISQ [6], [11], [15], [18], [19]. But there is practically no company that can afford unlimited resources to test and correct software faults in the real world. Consequently, the assumption of the ISQ model has to be properly modified and/or adjusted. In the following, an EFSQ model is proposed to analyze and discuss the processes of fault detection and removal.

III. EXTENDED FINITE SERVER QUEUEING MODEL

A. MODEL DESCRIPTIONS

The proposed EFSQ model for describing the activities of fault detection and correction is based on the following assumptions [6], [10], [20]:

1) The fault detection process follows an NHPP.
2) The software is subject to failures at random times, caused by the manifestation of remaining faults in the system.
3) The mean number of faults detected in the time interval $(t, t + \Delta t]$ is proportional to the mean number of remaining faults in the system. Further, all faults in a program are mutually independent from the failure detection point of view.
4) Fault detection activity continues while faults are being removed, and fault correction action does not affect the detection process.
5) The detected faults will be fixed by a *finite* number of debuggers (denoted c) in a group. The detected faults are placed in the queue according to a Poisson process with rate λ. The correction time of fault for each

debugger is exponentially distributed with the rate of μ. The queueing time and the correction time are mutually independent.

6) Fault removal time is non-negligible so that the number of removed faults may lag behind the total number of detected faults.

Based on assumptions (1)–(6), we can see that the EFSQ model is an $M/M/c$ queue model [21]. Suppose that an arbitrary software fault is found at time x; then the probability that the detected fault has been corrected by time t is equal to $G(t-x)$ and G(•) is the *Cumulative Distribution Function* (CDF) of service (failure correction) time [6], [10], [16], [20]. Under this condition, we let p be the probability that a fault is detected at time x, and will be completely corrected in $[x, t]$. From total probability theorem, we obtain [10]:

$$\begin{aligned} &P\{\text{Time required for complete removal} \\ &\quad \le t-x \cap \text{ fault detected at } x\} \\ &= P\{\text{Time required for complete removal} \\ &\quad \le t-x | \text{ fault detected at } x\}\, P\{\text{fault detected at } x\} \end{aligned}$$

The probability that a fault is detected at time x is given by

$$P\{\text{fault is detected at time } x\} = \frac{m_d'(x)}{m_d(t)}, \tag{3}$$

where $m_d(t)$ is the cumulative number of detected faults at time t and $m_d'(x)$ is the derivative of $m_d(x)$ with respect to x. In this case, we have [6], [10]:

$$\begin{aligned} p &= \int_0^t P\{\text{Time required for complete removal} \\ &\qquad \le t-x \cap \text{ fault detected at } x\} \\ &= \int_0^t P\{\text{Time required for complete removal} \\ &\qquad \le t-x | \text{ fault detected at } x\} \\ &\quad \times P\{\text{fault detected at } x\}\, dx = \int_0^t G(t-x)\frac{m_d'(x)}{m_d(t)}dx. \end{aligned} \tag{4}$$

The probability that a fault is detected at time x, and is not completely removed at time t is $q = 1-p$. Thus, we will have

$$P\{N_r(t) = i\} = \frac{[m_d(t)p]^i}{i!}\exp[-m_d(t)p], \tag{5}$$

where $N_r(t)$ denotes the counting process representing the cumulative number of faults removed (or corrected) by the time t. Because the expected number of faults removed at time t and the number of faults that were detected but have not been removed are independent of each other, the mean of $N_r(t)$ will be given by

$$m_r(t) = m_d(t) \times p. \tag{6}$$

Thus, we have $m_r(t) = \int_0^t G(t-x)m_d'(x)dx$ and the equation can be written as [6], [10]:

$$m_r(t) = \int_0^t g(t-x)m_d(x)dx. \tag{7}$$

In actuality, when a fault is detected and placed in the queue, the total time the fault stayed in the system is the time it stayed in the queue plus the time needed for being corrected and removed from the system. And if there are some debuggers still available when a fault enters the queueing system, the total time of a fault staying in the system is only the correction time, because it does not have to wait in queue for free debuggers. Thus, the probability that a fault has zero queueing time is given by [21]:

$$\begin{aligned} W_q(0) &= P\{\textit{queueing} \text{ time} = 0\} = P\{\le c-1 \text{ faults in system}\} \\ &= \sum_{n=0}^{c-1} p_n = 1 - \frac{r^c p_0}{c!(1-\rho)}, \end{aligned} \tag{8}$$

where P_n is the probability that there are n faults in the queue system and ρ is the average amount of faults coming to each debugger per unit time ($=\lambda/c\mu$). We can also obtain the average length of queue L_q, that is, the average number of faults waiting in the queue [21]:

$$L_q = \sum_{n=c+1}^{\infty}(n-c)p_n = \sum_{n=c+1}^{\infty}(n-c)\frac{r^n}{c^{n-c}}p_0 \tag{9}$$

Applying Little's formula on Eq. (9), we can get the average time that a fault has to wait in the queue [20], [21]:

$$W_q = L_q/\lambda \tag{10}$$

Here we separated the total time required by a fault into two circumstances, namely, those faults having zero queueing time with probability $W_q(0)$, and those with the total time being a queue delay plus the correction time with probability $1 - W_q(0)$. For simplicity, we use $T_{total}(t)$ to represent the CDF of the total time:

$$T_{total}(t) = T_1(t) + T_2(t), \tag{11}$$

$$T_1(t) = P\{\text{queueing time} = 0\}(\text{CDF of correction time } T_s), \tag{12}$$

and

$$\begin{aligned} T_2(t) &= P\{\text{queueing time} > 0\} \\ &\quad \times \big(\text{CDF of correction time } T_s + \text{queueing time } T_q\big). \end{aligned} \tag{13}$$

Notice that the sum distribution of two independent random variables T_s and T_q is the convolution of their individual distributions [21]–[23].

It's worth noting that Gokhale and Mullen [8] studied that the service time of a detected fault for an engineer is exponentially distributed, irrespective of the defect severity. Kapur et al. [15] and Xie et al. [24] also reported that it is more practical to use the exponential distribution for fault removal times. Thus it is assumed that the CDF of correction time T_s is identical to the exponential distribution with mean $1/\mu$ according to assumption (5). The distribution of T_q has a mean of $1/(c\mu - \lambda)$ [21]. Let $F(t)$ and $G(t)$ be the CDF

of correction time and queueing time, respectively. Thus we have

$$F(t) = 1 - \exp[-\mu t] \tag{14}$$

and

$$G(t) = 1 - \exp[-(c\mu - \lambda)t] \tag{15}$$

Here the Laplace transform will be used to get the convolution of $F(t)$ and $G(t)$. According to the convolution theorem [21]–[23], we have

$$G^*(s) = L\left(F(t) * G(t)\right) = L(F(t)) \times L(G(t)). \tag{16}$$

That is, $G^*(s) = \left(\frac{\mu}{s(s+\mu)}\right)\left(\frac{c\mu-\lambda}{s(s+c\mu-\lambda)}\right)$. By the partial fraction expansion, the above equation can be rewritten as:

$$G^*(s) = \left(\frac{c(1-\rho)}{c(1-\rho)-1}\right)\left(\frac{\mu}{s(s+\mu)}\right) - \left(\frac{1}{c(1-\rho)-1}\right)\left(\frac{c\mu-\lambda}{s(s+c\mu-\lambda)}\right) \tag{17}$$

Then we have

$$(F * G)(t) = \left(\frac{c(1-\rho)}{c(1-\rho)-1}\right)(1-\exp[-\mu t]) - \left(\frac{1}{c(1-\rho)-1}\right)(1-\exp[-(c\mu-\lambda)t]). \tag{18}$$

Substituting Eqs. (14), (15), and (18) into Eq. (11), we obtain [21]:

$$\begin{aligned} T_{total}(t) &= W_q(0)(1-\exp[-\mu t]) + \left(1 - W_q(0)\right) \\ &\quad \times \left\{ \begin{array}{l} \frac{c(1-\rho)}{c(1-\rho)-1}(1-\exp[-\mu t]) - \frac{1}{c(1-\rho)-1} \\ \times(1-\exp[-(c\mu-\lambda)t]) \end{array} \right\} \\ &= \frac{c(1-\rho)-W_q(0)}{c(1-\rho)-1}(1-\exp[-\mu t]) \\ &\quad - \frac{1-W_q(0)}{c(1-\rho)-1}(1-\exp[-(c\mu-\lambda)t]) \end{aligned} \tag{19}$$

Continuing from Eq. (4) we have

$$p = \int_0^t T_{total}(t-x)\frac{m_d'(x)}{m_d(t)}dx. \tag{20}$$

Let $N_o(t)$ be the number of open remaining faults (i.e., detected but not corrected faults) at time t, apply the binomial theorem, and regard the n detections as independent trials [6], [16]. In this case, we have

$$P\{N_r(t)=i, N_o(t)=n-i|N_d(t)=n\} = \frac{n!}{i!(n-i)!}p^i q^{n-i}. \tag{21}$$

That is, $P\{N_r(t)=i, N_o(t)=n-i\} = P\{N_r(t)=i, N_o(t)=n-i|N_d(t)=n\} \times P\{N_d(t)=n\}$

$$= \frac{[m_d(t)p]^i}{i!}\exp[-m_d(t)p]\frac{[m_d(t)q]^{n-i}}{(n-i)!}\exp[-m_d(t)q]. \tag{22}$$

Since $N_o(t)$ and $N_r(t)$ are independent of each other, by total probability theorem [6], [10], we obtain

$$P\{N_o(t)=n-i\} = \frac{[m_d(t)q]^{n-i}}{(n-i)!}\exp[-m_d(t)q]. \tag{23}$$

And further, from Eq. (6), we have

$$\begin{aligned} m_r(t) &= m_d(t)\int_0^t T_{total}(t-x)\frac{m_d'(x)}{m_d(t)}dx \\ &= \int_0^t T_{total}(t-x)m_d'(x)dx. \end{aligned} \tag{24}$$

Similarly, the MVF of open-remaining faults $m_o(t)$ is

$$m_o(t) = \int_0^t [1 - T_{total}(t-x)]\, m_d'(x)dx. \tag{25}$$

B. RELATIONSHIP TO OTHER SRGMs

Here the relationship between the proposed EFSQ model and other SRGMs will be presented and discussed. First, from Eq. (20), Eq. (5) can be rewritten as

$$P\{N_r(t)=i\} = \frac{\left[\int_0^t T_{total}(t-x)dm_d(x)\right]^i}{i!}\exp\left[-\int_0^t T_{total}(t-x)dm_d(x)\right]. \tag{26}$$

Thus we can determine the time-dependent behavior of the fault removal process by utilizing $T_{total}(t-x)$ and $m_d(x)$. It can also be found that some existing classical SRGMs can be derived from Eq. (26) when the number of debuggers c approaches infinity. For example, in Eq. (19), if c approaches infinity, we will have [21]:

$$\begin{aligned} \lim_{c\to\infty} T_{total}(t) &= \lim_{c\to\infty}\{W_q(0)\,(1-\exp[-\mu t]) \\ &\quad + [1-W_q(0)][\frac{c(1-\rho)}{c(1-\rho)-1}(1-\exp[-\mu t]) \\ &\quad - \frac{1}{c(1-\rho)-1} \times (1-\exp[-(c\mu-\lambda)t])]\} \\ &= \lim_{c\to\infty}\left\{W_q(0)\right\}\lim_{c\to\infty}\{1-\exp[-\mu t]\} \\ &\quad + \lim_{c\to\infty}\left\{1-W_q(0)\right\} \\ &\quad \times \lim_{c\to\infty}\{[\frac{c(1-\rho)}{c(1-\rho)-1}(1-\exp[-\mu t]) \\ &\quad - \frac{1}{c(1-\rho)-1}(1-\exp[-(c\mu-\lambda)t])]\}. \end{aligned} \tag{27}$$

In addition, it can be seen from Eq. (19) that $T_{total}(t)$ is composed of $W_q(0)$ and $1 - W_q(0)$. According to the definition of $W_q(0)$, there would be no need to wait in line when there are infinite debuggers available. Therefore, the probability that a fault does not have to wait in the queue is 1 when there are infinite debuggers [21]. Consequently, we obtain $\lim_{c\to\infty}\left(W_q(0)\right) = 1$. We can further calculate the

distribution of total time a fault has spent in the queueing system when c approaches infinity. That is,

$$\lim_{c\to\infty} T_{total}(t) = 1 \times (1 - \exp[-\mu t]) + 0 \times \left(\frac{c(1-\rho) \times (1 - \exp[-\mu t])}{c(1-\rho) - 1} - \frac{(1 - \exp[-(c\mu - \lambda)t])}{c(1-\rho) - 1} \right) = 1 - \exp[-\mu t]. \quad (28)$$

We can see that Eq. (28) is the distribution of the exponential correction time [1]. When infinite debuggers are available, a fault can be fixed whenever it enters the queueing system and the only time it has to wait is the exponential correction time until it's fixed.

If the fault detection process is described by the Yamada delayed S-shaped model [4]:

$$m_d(t) = a\,(1 - (1 + bt)\exp[-bt]) \quad (29)$$

(where a is the expected number of total faults in the software, and b is the fault detection rate), we can substitute Eqs. (28) and (29) into Eq. (26), and the MVF of Eq. (26) can be generalized as

$$m_r(t) = a\left(1 - (1 - bt + \frac{b^2(bt - \mu t + 1)}{(b-\mu)^2})\exp[-bt] - \frac{b^2}{(b-\mu)^2}\exp[-\mu t]\right). \quad (30)$$

We can acquire the following equation, assuming that $b = \mu$:

$$m_r(t) = a\,(1 - (1/2)\exp[-bt](2 + bt(2 + bt))) \quad (31)$$

On the other hand, if the detection process is described by the Goel-Okumoto model [1], [4], that is,

$$m_d(t) = a(1 - \exp[-bt]). \quad (32)$$

we substitute both Eq. (28) and Eq. (32) into Eq (26), and obtain:

$$m_r(t) = a\,(1 - 1/(b-\mu))\,(b\exp[-\mu t] - \mu\exp[-bt]) \quad (33)$$

Note that if $b = \mu$, Eq. (33) will become

$$m_r(t) = a\,(1 - (1 + \mu t)\exp[-\mu t])\,. \quad (34)$$

And it is the classical Yamada delayed S-shaped SRGM [25].

IV. DATA ANALYSIS AND NUMERICAL EXAMPLE

A. DESCRIPTIONS OF DATA SETS

In this paper, we will validate the performance of the EFSQ proposed model based on two data sets from real software projects. The first data set (DS1) was from System T1 of the Rome Air Development Center project reported by Musa [16], [26], and was collected weekly from each project member. The system T1 was used for a real-time command and control application and, the size of the software is approximately 21,700 object instructions. It took twenty-one weeks and nine programmers to complete the test. During the test phase, about 25.3 CPU hours were consumed and 136 software faults were removed. The second data set (DS2) was from an US Air Force software system reported by Goel and Yang in [6] and [27]. Over the course of 86 months, 4,538 faults were detected and 4,312 faults were corrected eventually.

Table 1 illustrates the selected models for performance comparison and the classification scheme. It is noted that the selected MVFs of describing the fault detection process are the Yamada delayed S-shaped (denoted DSS) model, and the inflected S-shaped (denoted ISS) model [1], [4]. Additionally, the MVFs of describing the fault removal process are the proposed EFSQ models (i.e., Eq. (24)) and the Yamada ISQ models (i.e., Eq. (2)). Here the selected models will be divided into two groups. The classification is mainly based on the MVF we used in the phase of fault detection (i.e., $m_d(t)$). Thus the DSS model, our proposed EFSQ model #1, and the ISQ model #1 will be included in Group A. Group B is made up of the ISS model, our proposed EFSQ model #2, and the ISQ model #2.

B. COMPARISON CRITERIA

Kanoun et al. [28] reported that a model can be analyzed according to its *Retrodictive* and *Predictive* capabilities. But a model that performs well according to one criterion could perform poorly according to another. In order to check the performance of proposed model, make a fairly comprehensive comparison with other models, and avoid bias, we use some criteria as follows.

1) The *Relative Error* (RE) is defined by [16], [26], [29]:

$$RE = \frac{m(t_i) - m_i}{m_i}, \quad (35)$$

TABLE 1. Classification scheme of selected models.

Models (Group A)	Descriptions
Yamada delay S-shaped model (DSS model)	$m_d(t) = m_r(t) = a(1-(1+bt)\exp[-bt])$
Proposed EFSQ model #1	$m_d(t)$: DSS model; $m_r(t)$: Eq. (24)
Yamada ISQ model #1	$m_d(t)$: DSS model; $m_r(t)$: Eq. (2)
Models (Group B)	**Descriptions**
Inflection S-shaped model (ISS model)	$m_d(t) = m_r(t) = a(1-\exp[-bt])\,/\,(1+\psi\exp[-bt])$
Proposed EFSQ model #2	$m_d(t)$: ISS model; $m_r(t)$: Eq. (24)
Yamada ISQ model #2	$m_d(t)$: ISS model; $m_r(t)$: Eq. (2)

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 estimated by a model. Positive values of error indicate overestimation; negative values of error indicate underestimation.

2) The *Mean Square Error* (MSE) is used to judge the *Retrodictive* ability and is typically defined as [16], [29], [30]

$$MSE = \frac{1}{n-\theta}\sum_{i=1}^{n}(m(t_i) - m_i,)^2 \tag{36}$$

where n is the size of the selected data set and θ is the degree of freedom. A smaller MSE indicates a smaller fitting error, and better performance.

3) The *Mean Absolute Error* (MAE) is typically defined as follows [31], [32]:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|m_i - m(t_i)|. \tag{37}$$

The MAE indicates the magnitude of the average error and it is a linear score, which means that all the individual differences are weighted equally in the average.

4) The *variance* is a measure of how far a set of numbers is spread out, and it is defined as [33]:

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

and

$$Bias = \sum_{i=1}^{n}\frac{m(t_i) - m_i}{n}. \tag{38}$$

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

5) The *Coefficient of Determination* (R^2) is defined as [16], [29]:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(m(t_i) - m_i)^2}{\sum_{i=1}^{n}(m_i - \bar{m})^2}$$

and

$$\bar{m} = \frac{\sum_{i=1}^{n} m_i}{m_i}. \tag{39}$$

R^2 denotes the percentage of total variation about the mean accounted for by the fitted curve. A larger value indicates a better fit.

6) The *Theil's U Statistic* (TS) is a well-known econometrics inequality measure. In this paper, the *Theil's U Statistics* will be calculated and presented in both of its specifications, which are labeled U1 and U2, respectively [34]–[36]:

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

and

$$U2 = \sqrt{\frac{\sum_{i=1}^{n-1}\left(\frac{m(t_{i+1}) - m_{i+1}}{m_i}\right)^2}{\sum_{i=1}^{n-1}\left(\frac{m_{i+1} - m_i}{m_i}\right)^2}} \tag{40}$$

A low value of Ul and U2 indicates a more accurate predictive capability of the model.

7) The *goodness-of-fit* statistical test is commonly used to determine whether the data from a sample comes from a population with a specific distribution according to a specific hypothesis [1], [4], [16], [29], [30], [37]. In the past, both the *Kolmogorov-Smirnov* (K-S) test and the *Chi-Square* (χ^2) test have been widely used and recommended for the goodness-of-fit testing and the post-model application for software reliability growth models. The χ^2 test is typically defined by [4], [20]:

$$\chi^2 = \sum_{i=1}^{n}\frac{(m_i - m(t_i))^2}{m(t_i)}. \tag{41}$$

Notice that the null and the alternative hypotheses are H: *the data follow the specified distribution*, and H_1: *the data do not follow the specified distribution*. The hypothesis regarding the distributional form is rejected at the chosen significance level α if the result of (41) is greater than the critical value defined as $\chi^2_{(1-\alpha,n-1)}$ with $n-1$ degree of freedom. In this paper, the χ^2 test will be compared with the critical value for the 80% significance level. We will use the arrow ($\uparrow$, $\downarrow$) in the following experiments to indicate whether or not the selected model fits the actual data.

8) The *Kolmogorov-Distance* (KD) is defined as [4], [20], [37]:

$$D_K = Sup_x|F^*(x) - F(x)|, \tag{42}$$

where k is the sample size $F^*(x)$ is the normalized observed cumulative distribution at the x-th time point, and $F(x)$ is the expected cumulative distribution at the x-th time point, based on the model.

9) *Prediction at level l*. In a set of n software components, let k be the number of components whose mean magnitude of relative error is less than or equal to l. Then PRED(l) is defined by [29], [30], [38]:

$$\text{PRED}(l) = \frac{k}{n}. \tag{43}$$

PRED(l) measures the percentage of estimated values that are within l% of their actual values.

TABLE 2. Performance of DSS and ISS models for the detected faults of DS1.

Models	MSE	MAE	TS-U1	TS-U2	R^2	Var.	KD
DSS model	374.97	14.93	0.12	6.40	0.86	11.93	0.21
ISS model	30.25	3.80	0.04	0.65	0.99	3.91	0.07

TABLE 3. Parameter estimates of selected models for the corrected faults of DS1.

Models	a	b	ψ	μ	λ	c	ρ
DSS model	504.44	0.040					
ISS model	155.21	0.340	184.9				
Proposed EFSQ model #1	603.48	0.038		0.876	6.06	7.20	0.96
Proposed EFSQ model #2	155.22	0.340	184.9	0.933	6.82	7.44	0.98
Yamada ISQ model #1	603.48	0.038		6.46			
Yamada ISQ model #2	155.22	0.340	184.9	6.53			

TABLE 4. Comparison results for the corrected faults of DS1.

Models (Group A)	MSE	MAE	KD	TS-U1	TS-U2	R^2	χ^2	RE	Var.
DSS model	286.15	13.06	0.22	0.12	3.44	0.90	103.68↓	-0.16	16.81
Proposed model #1	236.42	11.99	0.22	0.11	3.72	0.91	99.86↓	-0.10	15.28
Yamada ISQ model #1	236.65	11.99	0.22	0.11	3.73	0.91	100.11↓	0.10	15.30
Models (Group B)	**MSE**	**MAE**	**KD**	**TS-U1**	**TS-U2**	**R^2**	**χ^2**	**RE**	**Var.**
ISS model	101.99	7.79	0.13	0.07	1.93	0.96	229.35↓	-0.04	10.04
Proposed model #2	22.97	3.21	0.07	0.03	0.98	0.99	14.68↑	-0.01	4.76
Yamada ISQ model #2	25.38	3.44	0.07	0.04	0.98	0.99	14.95↑	0.01	5.00

C. PERFORMANCE ANALYSIS AND DISCUSSION

In general, the *Least Square Estimation* (LSE) and the *Maximum Likelihood Estimation* (MLE) are used for the parameter estimation of models [4], [16], [26], [27], [30]. It is noted that sometimes MLE tends to be biased [16], or may lead one to difficulties [39], but LSE can produce unbiased results [40]. Thus in this paper, LSE is mainly used to estimate the model parameters for DS1 and DS2.

1) DS1

First, using the detected faults of DS1, we obtain $a = 237.19$ and $b = 963 \times 10^{-2}$ for the DSS model, and $a = 154$, $b = 0.35$, and $\psi = 173.0$ for the ISS model. It is instructive to examine the estimated value of the inflection factor (ψ) for the ISS model in detail. According to the definition of the ISS model, it can be found that $\psi = (1 - \varphi)/\varphi$ and φ is the inflection rate [4]. The inflection rate can be interpreted as the ratio of the number of detectable faults to the total number of faults in the program. Since the estimated value of ψ is 173, thus we have $\varphi = 0.57 \times 10^{-2}$ for the fault detection data of DS1, indicating that only a few faults were detectable at the beginning and faults rapidly became detectable thereafter. That is, the observed software reliability growth could be equivalent to the S-shaped curve [1]. Consequently, the DSS model and the ISS model are suitable candidate models for this data set.

Table 2 lists the performance of the DSS and ISS models for the detected faults of DS1 in terms of MSE, MAE, TS-U1, TS-U2, R^2, variance, and KD. It can be seen that the DSS and ISS models fit the failure data of fault detection better. Thus they can be used for the subsequent investigation and modeling of the fault removal process. Table 3 shows the estimated parameters of all selected models for the corrected faults of DS1. As seen from Table 3, the estimated value of c for the proposed EFSQ models #1 and #2 is 7.2 and 7.44, respectively. Musa argued that available debuggers would not always be fully employed during the failure-correction-personnel-limited period due to the inequality in load among the debuggers [16].

Table 4 gives the performance comparisons in terms of MSE, MAE, KD, TS-U1, TS-U2, R^2, χ^2, and variance. We can see from Table 4 that in Group A, the proposed EFSQ model #1 gives the lowest MSE, MAE, KD, TS-U1, and TS-U2 compared to the DSS model and the Yamada ISQ model #1 for the corrected failure data of DS1. Moreover, the proposed EFSQ model #1 provides the larger R^2 value compared with the DSS model.

On the other hand, we can also see from Table 4 that for Group B, the proposed EFSQ model #2 gives the lowest MSE, MAE, KD, TS-U1, and TS-U2 compared to the ISS model and the Yamada ISQ model #2. In addition, the R^2 value of the proposed EFSQ model #2 is also larger than that of the ISS model in Group B. Note that Table 4 also shows the RE values for all selected models. For example, for 100% of the total corrected data (i.e., at the end of testing in the 21st week), from Eq. (24), i.e., the MVF of the proposed

EFSQ model #2, we obtain $m_r(21)$=134.55. Thus we can compute the RE as $(134.55 - 136)/136 = -0.01$ from Eq. (35). In other words, the proposed EFSQ model #2 underestimates by 1 percent at the end of testing. But it also has to be noted that the ISS model underestimates by 4 percent, and the Yamada ISQ model #2 overestimates by 1 percent.

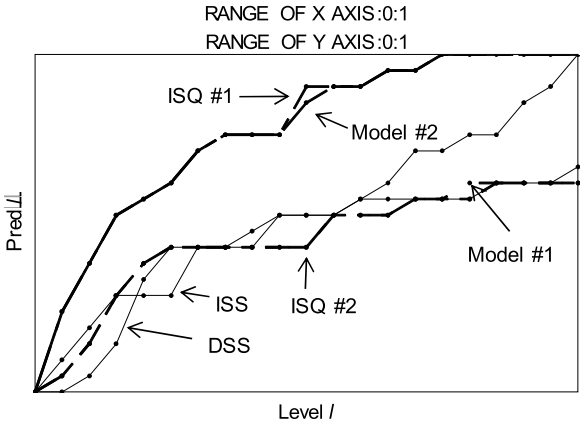

FIGURE 1. Pred(l) curves of the corrected faults (DS1).

Finally, Fig. 1 plots the PRED(l) curves (taken with respect to different l levels) of all selected models. In actuality, the plots of PRED(l) are useful tools for drawing conclusions about the relative predictive capability of models [29], [30]. If one model is better than another model in terms of this criterion, its PRED curve would lie on top of that for another model. We can see from Fig. 1 that the PRED (0.25) values for the DSS model, the proposed EFSQ model #1, the Yamada ISQ model #1, the ISS model, the proposed EFSQ model #2, and the Yamada ISQ model #2 are 0.43, 0.43, 0.62, 0.29, 0.62, 0.43, respectively. Because there are totally 21-week tests in DS1 (i.e., the x axis), the number of estimates that are within 25% of actual values are 9, 9, 13, 6, and 9, respectively.

At $l = 0.25$, we can find that more than 60% of the estimates from the proposed EFSQ model #2 are within 25% of the actual values. The corresponding percentages for the ISS model and the Yamada ISQ model #2 are 29% and 43%, respectively. It is also interesting to note that the curves for the DSS model and the Yamada ISQ model #2 cross at $l = 0.25$. An examination of Fig. 1 suggests that the proposed EFSQ model #2 performs well at different l levels. Overall, from these figures, tables, and comparison criteria, the proposed EFSQ model #2 gives a better fit to the DS1, and predicts future behavior well.

2) DS2

Similarly, using the detected faults of DS2, we obtain $a = 14219.19$ and $b = 135 \times 10^{-2}$ for the DSS model, and $a = 4610.05$, $b = 0.1$, and $\psi = 196.86$ for the ISS model. It is also noticed that the estimated value of the inflection factor for the ISS model is 196.86. Thus the inflection rate $\varphi = 0.51 \times 10^{-2}$, and this indicates an S-shaped growth curve. Both the DSS model and the ISS model still are in the set of candidate models. Table 5 shows the performance of the DSS and ISS models for the detected faults of DS2 in terms of MSE, MAE, TS-U1, TS-U2, R^2, variance, and KD. It can be seen that both the DSS and the ISS models provide a good fit for the fault detection data of DS2, as illustrated in Table 5. Thus they still will be used for describing the fault removal process of DS2.

The estimated parameters of selected models for DS2 are presented in Table 6. As seen from Table 6, the estimated number of debuggers for the proposed EFSQ models #1 and #2 is 216.32 and 198.2, respectively. Note that we may not be able to find out the actual number of debuggers for DS2 from Yang's past studies in [6] and [27]. But Yang [6] once reported that DS2 was collected from an US Air Force software system, and the total amount of testing time is 86 months [6]. There were 4,538 faults that were detected and 4,312 faults that were corrected eventually. It can be inferred that large-scale development teams were responsible for software development and testing. Thus it

TABLE 5. Performance of DSS and ISS models for the detected faults of DS2.

Models	MSE	MAE	TS-U1	TS-U2	R^2	Var.	KD
DSS model	1.18×10^5	280.97	0.07	16.23	0.95	331.05	0.137
ISS model	0.90×10^5	75.05	0.02	4.88	0.99	92.95	0.049

TABLE 6. Parameter estimates of selected models for the corrected faults (DS2).

Models	a	b	ψ	μ	λ	c	ρ
DSS model	13581.2	1.17×10^{-2}					
ISS model	4704.3	0.1	210.2				
Proposed EFSQ model #1	14629.6	1.17×10^{-2}		0.29	59.63	216.32	0.95
Proposed EFSQ model #2	4713.33	0.1	210.26	0.27	52.5	198.2	0.98
Yamada ISQ model #1	14629.6	1.17×10^{-2}		61.9			
Yamada ISQ model #2	4713.33	0.1	210.26	56.1			

TABLE 7. Comparison results for the corrected faults (DS2).

Models (Group A)	MSE	MAE	KD	TS-U1	TS-U2	R^2	χ^2	RE	Var.
DSS model	1.64×10^5	332.03	0.18	0.09	32.97	0.94	$13.54\times10^3\downarrow$	-0.09	404.57
Proposed model #1	1.53×10^5	350.36	0.24	0.09	30.60	0.94	$13.00\times10^3\downarrow$	-0.06	391.00
Yamada ISQ model #1	1.53×10^5	332.04	0.18	0.09	32.88	0.94	$13.53\times10^3\downarrow$	-0.06	391.00
Models (Group B)	**MSE**	**MAE**	**KD**	**TS-U1**	**TS-U2**	R^2	χ^2	**RE**	**Var.**
ISS model	1.16×10^5	80.63	0.06	0.03	5.34	0.99	$11.95\times10^2\downarrow$	-0.03	107.78
Proposed model #2	1.20×10^5	78.62	0.05	0.03	5.32	0.99	$11.88\times10^2\downarrow$	-0.03	109.87
Yamada ISQ model #2	1.53×10^5	322.99	0.11	0.08	12.73	0.94	$75.34\times10^2\downarrow$	-0.05	390.90

should be reasonable that there were nearly two hundred developers for this software system.

Table 7 gives the results of model comparisons in terms of MSE, MAE, KD, TS-U1, TS-U2, $R^2\chi^2$, and variance. As for the results presented in Table 7, the proposed EFSQ model #1 gives the lowest TS-U1, TS-U2, and χ^2 compared to the DSS model and the Yamada ISQ model #1. As seen from Table 7, the computed RE values are -0.09, -0.06, -0.06 for the DSS model, the proposed EFSQ model #1, and the Yamada ISQ #1 model, respectively. That is, for 100% of the detected faults of DS2, both the proposed EFSQ model #1 and the Yamada ISQ #1 model underestimate by 6 percent, but the DSS model underestimates by 9 percent. Additionally, the RE values for the ISS model, the proposed EFSQ model #2, and the Yamada ISQ model #2 are -0.03, -0.03, and -0.05, respectively.

We can also see that the proposed EFSQ model #2 gives the lowest MAE, KD, χ^2, TS-U1, and TS-U2 in Group B, as shown in Table 7. Although the proposed EFSQ models #1 and #2 don't provide the smallest MSE in Groups A and B, the difference is small. Additionally, the R^2 and PRED (0.25) values of the proposed EFSQ models #1 and #2 are larger than those of the ISS model, the DSS model and the Yamada ISQ models #1 and #2. Finally, Fig. 2 also shows the PRED(l) plots of all models. It can be found that the proposed EFSQ model #2 performs well at different l levels. Overall, the proposed EFSQ model #2 performs reasonably well on DS2.

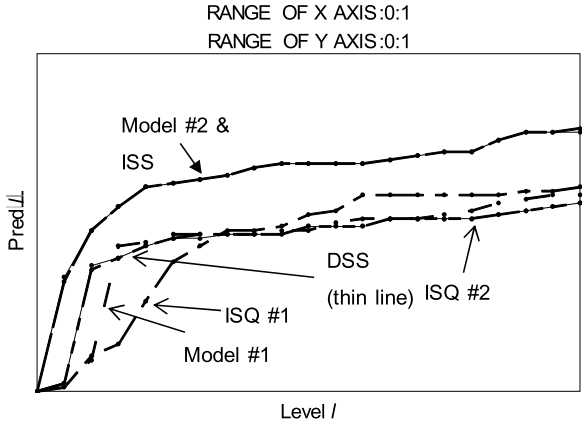

FIGURE 2. Pred(l) curves of the corrected faults (DS2).

D. DISCUSSION OF VALIDITY CONSIDERATIONS

In this section, the threats to both internal and external validity of this paper will be briefly discussed [2], [41]. There are some factors that may be able to affect the internal validity. First, the threats to the internal validity are the collection process of software failure data we used. Additionally, this could be a risk for the selected data sets that were incomplete or partial, and therefore insufficient for quantitative modeling. In practice, user reported defects may not always be well documented or testers could report faults informally (keeping them out of the defect tracking system).

On the other hand, the threat to internal validity has also to be concerned with the estimated values of model parameter. In practice, a prediction model that works well for one data may not guarantee to perform well for the other data. In order to eliminate bias and control for this threat, other methods can also be used for the parameter estimation of selected models. Another threat to internal validity is the number of candidate models when comparing the predicted value to the real value.

Finally, in order to minimize the threats to external validity of the experiments, two failure data collected from different projects were used. Note that DS1 has been widely cited by many researches in the field of software reliability engineering. In principle, if we propose a new model, it should be compared with other existing models. All models are needed to apply to the same software failure data and the experimental results should be provided to show whether the proposed new model fits better than the other models or not. Thus the threats to external validity should be minimal.

V. APPLICATIONS FOR SOFTWARE PROJECT MANAGEMENT

In this section, we will show how to apply the proposed EFSQ model to project management and control in software development. Due to space limitations, here we only select DS1 and the proposed EFSQ model #2 to illustrate the suggested project management applications. Similar analyses and discussions can be applied to the other models or DS2.

First, we can see from Table 3 that the estimated fault detection and correction rates of the proposed EFSQ model #2 are $\lambda = 6.82$ faults/week and $c\mu = 6.94(= 7.44 \times 0.933)$

TABLE 8. The statistical results of using various M/M/c models for DS1.

Model Parameters	M/M/1	M/M/2	M/M/3	M/M/4	M/M/5	M/M/6	M/M/7	M/M/8	M/M/9
L_q	61.35	64.73	53.15	61.35	64.63	64.29	57.57	61.36	64.63
W_q	81.26	42.84	23.52	20.32	17.11	14.18	10.9	10.16	9.51
$T_{\text{mean response time}}$	93.00	48.71	27.43	23.25	19.45	16.14	12.58	11.63	10.81

faults/week, respectively. Thus we have $\rho = 0.98$ and this means that project managers nearly run out of the resources of the software development team. Obviously, the original scheduling would not be able to match the actual timing if feature (or scope) creeps in or unexpected events happen, such as if the developers were to leave, new developers were to be added too frequently, etc. One of the mitigation strategies is to delay the release schedule.

Musa et al. [16] once reported that it took 21 weeks and nine programmers to test the system for DS1. It can also be found that the number of detected faults is one after the test of 20 weeks, and this number is relatively low. Consequently, if we would like to extend the release deadline for DS1 to 26 weeks, i.e., a five-week deadline extension, the modified fault detection and correction rates can be computed by $\lambda' = (6.82 \times 21)/(21 + 5) = 5.51$ faults/week and $\mu' = (6.94 \times 21)/(21 + 5) = 5.61$ faults/ week [13], respectively. But we still can find that the modified value of ρ is less than 1 and all faults can be corrected by the time the software is released. In this case, the maximum percentages of staff resources can be reduced by $100 \times (6.94 - 5.61)/6.94 = 19.16\%$. That is, if originally nine developers were planned and were required to test the system, we can make some reasonable adjustments on the personnel arrangement.

Finally, the statistical results of using various $M/M/c$ models for the corrected faults of DS1 are illustrated in Table 8. For instance, from Eqs (9) and (10), we obtain $L_q = \sum_{n=10}^{\infty} (n-9) \times 8.87^n \times (1.47 \times 10^{-5})/(9^{n-9}) = 64.6$ faults and $W_q = L_q/\lambda = 64.6/6.8 = 9.51$ weeks if there are nine developers. This indicates that an average of 64.6 faults was waiting in the queue, and the average time required for a fault to stay in the queue was 9.5 weeks. Notice that the mean response time that a detected fault spends in the queue until it is fixed can also be calculated. It can be found that both the average waiting time and the mean response time get high when there are fewer debuggers. In such a circumstance, even a minor change may lead to the instability of the process. To avoid this, more resources are required and have to be added to improve the response time. Since the debugging process described in this paper is a finite queueing system, we will be able to increase the overall debugging rate by increasing personnel resources. Additionally, extending the schedule may also be helpful when it comes to saving personnel resources, but it may have no effect on shortening the response time.

VI. CONCLUSION

The fault detection and removal processes play important roles in software development and reliability assessment. Most of classical SRGMs have assumed that the fault removal time can be ignored. In actuality, this assumption may not always be true since the developers indeed need time to analyze the root causes of the failure and correct them later. Although some studies have tried to take the fault correction time into consideration and also showed that the debugging can be described by the ISQ model, there is no software developing team that owns infinite (personnel) resources in the debugging process. In this paper, we thoroughly applied the queueing theory to jointly investigate the fault removal process, considering fault correction time and finite debugging resources. We selected and utilized some classical SRGMs as the candidate models to depict the processes of software testing and debugging. We also showed and illustrated that the ISQ model is a special case of the proposed EFSQ model. Also note that the proposed EFSQ model can be reduced to classical SRGMs under some circumstances Experiments are performed based on real software failure data and the experimental results demonstrate that the proposed EFSQ models have a better goodness-of-fit and predict the future behavior well.

Finally, we also presented how to use the proposed EFSQ model from a project management perspective. In actuality knowing the status of the average number of faults that are waiting in the queue and the mean response time can greatly assist developers in efficiently setting up a reasonable schedule. Such information can also provide project management with flexibility in resource allocation and handling unexpected events. Our proposed model and method provide an effective foundation for managing the necessary activities of software development, and can also be instrumental to various management decisions.

REFERENCES

[1] L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach* (Quantitative Software Engineering Series). New York, NY, USA: Wiley, 2006.

[2] C. Bird, V.-P. Ranganath, T. Zimmermann, N. Nagappan, and A. Zeller, "Extrinsic influence factors in software reliability: A study of 200,000 windows machines," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, Hyderabad, India, Jun. 2014, pp. 205–214.

[3] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Incorporating fault debugging activities into software reliability models: A simulation approach," *IEEE Trans. Rel.*, vol. 55, no. 2, pp. 281–292, Jun. 2006.

[4] M. R. Lyu, *Handbook of Software Reliability Engineering*. New York, NY, USA: McGraw-Hill, 1996.

[5] C.-Y. Huang and C.-T. Lin, "Software reliability analysis by considering fault dependency and debugging time lag," *IEEE Trans. Rel.*, vol. 55, no. 3, pp. 436–450, Sep. 2006.
[6] K.-Z. Yang, "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, 1996. [Online]. Available: http://surface.syr.edu/eecs_etd/189
[7] S. Balsamo, V. D. N. Personè, and P. Inverardi, "A review on queueing network models with finite capacity queues for software architectures performance prediction," *Perform. Eval.*, vol. 51, nos. 2–4, pp. 269–288, Feb. 2003.
[8] S. S. Gokhale and R. E. Mullen, "Queuing models for field defect resolution process," in *Proc. 17th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Raleigh, NC, USA, pp. 353–362, Nov. 2006.
[9] T. Dohi, S. Osaki, and K. S. Trivedi, "An infinite server queueing approach for describing software reliability growth: Unified modeling and estimation framework," in *Proc. 11th Asia-Pacific Softw. Eng. Conf. (APSEC)*, Busan, Korea, Nov./Dec. 2004, pp. 110–119.
[10] C.-Y. Huang and W.-C. Huang, "Software reliability analysis and measurement using finite and infinite server queueing models," *IEEE Trans. Rel.*, vol. 57, no. 1, pp. 192–203, Mar. 2008.
[11] S. Inoue and S. Yamada, "A software reliability growth modeling based on infinite server queueing theory," in *Proc. 9th ISSAT Int. Conf. Rel. Quality Design (QRD)*, Waikiki, HI, USA, Aug. 2003, pp. 305–309.
[12] G. Antoniol, A. Cimitile, G. A. Di Lucca, and M. Di Penta, "Assessing staffing needs for a software maintenance project through queuing simulation," *IEEE Trans. Softw. Eng.*, vol. 30, no. 1, pp. 43–58, Jan. 2004.
[13] B. Luong and D.-B. Liu, "Resource allocation model in software development," in *Proc. 47th IEEE Annu. Rel. Maintainability Symp. (RAMS)*, Philadelphia, PA, USA, Jan. 2001, pp. 213–218.
[14] Y. Malka and A. Ziv, "Design reliability-estimation through statistical analysis of bug discovery data," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 1998, pp. 644–649.
[15] P. K. Kapur, S. Anand, S. Inoue, and S. Yamada, "A unified approach for developing software reliability growth model using infinite server queuing model," *Int. J. Rel., Quality Safety Eng.*, vol. 17, no. 5, pp. 401–424, Oct. 2010.
[16] J. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. New York, NY, USA: McGraw-Hill, 1987.
[17] M. L. Shooman, *Software Engineering: Design, Reliability, and Management.* New York, NY, USA: McGraw-Hill, 1983.
[18] N. Schneidewind, "Software development queuing model," *J. Aerosp. Comput., Inf., Commun.*, vol. 7, no. 10, pp. 308–321, Oct. 2010.
[19] T. Dohi, K. Shibata, K. Rinsaka, and H. Okamura, "Quantifying software maintainability based on a fault-detection/correction model," in *Proc. 13th IEEE Int. Symp. Pacific Rim Dependable Comput. (PRDC)*, Melbourne, Vic., Australia, Dec. 2007, pp. 35–42.
[20] K. S. Trivedi, *Probability and Statistics With Reliability, Queuing, and Computer Science Applications*, 2nd ed. New York, NY, USA: Wiley, 2002.
[21] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, 4th ed. New York, NY, USA: Wiley, 2008.
[22] R. V. Hogg, J. McKean, and A. T. Craig, *Introduction to Mathematical Statistics*, 7th ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2012.
[23] Y. Katznelson, *An Introduction to Harmonic Analysis*, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[24] M. Xie, Q. P. Hu, Y. P. Wu, and S. H. Ng, "A study of the modeling and analysis of software fault-detection and fault-correction processes," *Quality Rel. Eng. Int.*, vol. 23, no. 4, pp. 459–470, Jun. 2007.
[25] C.-Y. Huang, S.-Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models," *IEEE Trans. Rel.*, vol. 56, no. 2, pp. 198–211, Jun. 2007.
[26] J. D. Musa and K. Okumoto, "A logarithmic poisson execution time model for software reliability measurement," in *Proc. 7th Int. Conf. Softw. Eng. (ICSE)*, Piscataway, NJ, USA, Mar. 1984, pp. 230–238.
[27] A. L. Goel and K.-Z. Yang, "Software reliability and readiness assessment based on the non-homogeneous Poisson process," *Adv. Comput.*, vol. 45, pp. 197–267, 1997.
[28] K. Kanoun, M. R. de Bastos Martini, and J. M. de Souza, "A method for software reliability analysis and prediction application to the TROPICO-R switching system," *IEEE Trans. Softw. Eng.*, vol. 17, no. 4, pp. 334–344, Apr. 1991.
[29] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Redwood City, CA, USA: Benjamin Cummings, 1986.
[30] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA, USA: PWS Pub., 1998.
[31] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Res.*, vol. 30, no. 1, pp. 79–82, Dec. 2005.
[32] (May 2008). *Department of Treasury, Government of Australia, Forecasting Accuracy of the ACT Budget Estimates Report*. [Online]. Available: http://www.treasury.act.gov.au/documents/Forecasting%20Accuracy%2020ACT%20Budget.pdf, accessed Dec. 2013.
[33] K. Pillai and V. S. Sukumaran Nair, "A model for software development effort and cost estimation," *IEEE Trans. Softw. Eng.*, vol. 23, no. 8, pp. 485–497, Aug. 1997.
[34] C.-Y. Huang and C.-T. Lin, "Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 283–288, Feb. 2010.
[35] P. L. Li, J. Herbsleb, and M. Shaw, "Forecasting field defect rates using a combined time-based and metrics-based approach: A case study of OpenBSD," in *Proc. 16th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Chicago, IL, USA, Nov. 2005, pp. 193–202.
[36] K. Holden, D. A. Peel, and J. L. Thompson, *Economic Forecasting: An Introduction*. Cambridge, U.K.: Cambridge Univ. Press, 1991.
[37] *NIST/SEMATECH e-Handbook of Statistical Methods*. [Online]. Available: http://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm, accessed Jun. 16, 2014.
[38] M. Shin and A. L. Goel, "Empirical data modeling in software engineering using radial basis functions," *IEEE Trans. Softw. Eng.*, vol. 26, no. 6, pp. 567–576, Jun. 2000.
[39] H. Okamura, T. Dohi, and K. S. Trivedi, "Markovian arrival process parameter estimation with group data," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1326–1339, Aug. 2009.
[40] M. Xie, "Software reliability models—Past, present and future," in *Recent Advances in Reliability Theory: Methodology, Practice, and Inference*, N. Limnios and M. Nikulin, Eds. Boston, MA, USA: Birkhauser, 2000, pp. 323–340.
[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. New York, NY, USA: Springer-Verlag, 2012.

CHIN-YU HUANG (M'05) is currently a Full Professor with the Department of Computer Science, National Tsing Hua University (NTHU), Hsinchu, Taiwan. He received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1994 and 2000, respectively. He was with the Bank of Taiwan, Taipei, from 1994 to 1999, and was a Senior Software Engineer with Taiwan Semiconductor Manufacturing Company, Ltd., Hsinchu, from 1999 to 2000. Before joining NTHU in 2003, he was the Division Chief with the Central Bank of China, Taipei. He was a recipient of the Ta-You Wu Memorial Award from the National Science Council of Taiwan in 2008. He also received an Honorable Mention Best Paper Award in the 2010 IEEE International Conference on Industrial Engineering and Engineering Management. In 2010, he was ranked as the 7th Place of Top Scholars in Systems and Software Engineering worldwide from 2004 to 2008 by the *Journal of Systems and Software* based on his research on software reliability, software testing, and software metrics. His research interests are in software reliability engineering, software testing, software metrics, and fault tree analysis. He has authored over 100 papers in these areas.

TZU-YU KUO received the B.E. degree (2011) in Department of Computer Science from National Tsing Hua University, Hsinchu, Taiwan and the M.E. degree (2013) in Department of Computer Science from National Tsing Hua University, Hsinchu, Taiwan. He is currently a software engineer in the Garmin Corporation, Taiwan. His research interests include software reliability and software measurement.