Interval Estimation Method for Decision Making in Wavelet-Based Software Reliability Assessment

Xiao XIAO\(^{(a)}\) and Tadashi DOHI\(^{(b)}\), Members

SUMMARY Recently, the wavelet-based estimation method has gradually been becoming popular as a new tool for software reliability assessment. The wavelet transform possesses both spatial and temporal resolution which makes the wavelet-based estimation method powerful in extracting necessary information from observed software fault data, in global and local points of view at the same time. This enables us to estimate the software reliability measures in higher accuracy. However, in the existing works, only the point estimation of the wavelet-based approach was focused, where the underlying stochastic process to describe the software-fault detection phenomena was modeled by a non-homogeneous Poisson process. In this paper, we propose an interval estimation method for the wavelet-based approach, aiming at taking account of uncertainty which was left out of consideration in point estimation. More specifically, we employ the simulation-based bootstrap method, and derive the confidence intervals of software reliability measures such as the software intensity function and the expected cumulative number of software faults. To this end, we extend the well-known thinning algorithm for the purpose of generating multiple sample data from one set of software-fault count data. The results of numerical analysis with real software fault data make it clear that, our proposal is a decision support method which enables the practitioners to do flexible decision making in software development project management.

key words: software reliability, uncertainty, interval estimation, Haar wavelet, nonparametric bootstrapping, non-homogeneous Poisson process, software-fault count data, thinning, prediction

1. Introduction

Nowadays, computer systems serve the key function in achieving highly complicated and safety-critical missions. Especially, the size and complexity of software, which is an important element in computer systems, have continued to increase. This trend makes the quality evaluation much more difficult than ever before. Usually, software is debugged during the testing phase of software development, so that the software reliability is improved over time as a result of detecting and removing software faults. But in fact, aside from the issue of cost, it is impossible to detect and remove all the faults in software. Hence the project leader is often faced with two options: (i) release the software to meet the budget and the delivery deadline, even if the software may not be reliable enough, or (ii) stay with software test some more to gain a higher reliability, even if this may lead to increased cost and delayed shipping time.

To support their decision making, many software reliability models (SRMs) that state the software fault-detection phenomenon have been proposed. Well-structured SRMs\cite{17,18,22,32} can provide reliable assessment results on the quantitative software reliability, which is defined as the probability that software system does not fail during a specified time period, meanwhile help to address many problems, such as development cost estimation, resource planning and software release scheduling. In the field of knowledge-based software reliability engineering, the non-homogeneous Poisson process (NHPP) based SRMs have gained much popularity because they can expertly integrate human experience into stochastic process\cite{17,22}. The NHPP-based SRM is governed by its mean value function, or equivalently by software intensity function. The quantitative software reliability can be calculated directly from the software intensity function, which is usually estimated from the observed software fault data. The tractability and satisfactory goodness-of-fit performance of the NHPP-based SRMs are the main reasons of their popularization.

With the inclusion of the representative ones proposed by Goel and Okumoto\cite{10}, Goel\cite{11}, Yamada \textit{et al.}\cite{33}, Abdel-Ghaly \textit{et al.}\cite{1}, and Zhao and Xie\cite{38}, almost all the NHPP-based SRMs are formulated as parametric models. That is, different parametric forms were assumed for the software intensity function in different NHPP-based SRMs. Note that, these assumptions are based on the knowledge of the software fault-detection rate, which is considered as constant, increasing or decreasing along testing time. However, the lesson learned from a huge number of empirical studies reported suggests that the best parametric NHPP-based SRM which can fit every type of software fault data does not exist. In other words, the past experiences or the knowledge we already have assuredly help in building a new model, but they often result in a system-dependent model that is specialized to specified software development project. This fact actually refers to the limitations of parametric models in practice and means that nonparametric approaches without assuming the parametric form should be used to describe the software debugging phenomenon.

Among the available nonparametric approaches in the literature to tackle this problem, the following methods have several theoretical and practical merits. Sofer and Miller\cite{23} used an elementary piecewise linear estimate of the software intensity function from the software-fault detection time data and proposed a smoothing technique by
means of quadratic programming. Gandy and Jensen [9] applied the well-known Aalen estimator to estimate the software intensity function. Although their estimator is statistically consistent, it is not feasible in many situations where the software products are tested by only one test team. Wang et al. [24] applied the kernel-based method to the NHPP-based SRM, where they focused on the local likelihood method with a local weighted log likelihood function. Recently, Xiao and Dohi [27]–[30] developed the wavelet shrinkage estimation. The wavelet transform decomposes the observed software fault data into time-frequency domain, so that the invisible knowledge involved in the observations can be extracted and presented in the form of frequency. This is one of the main reasons of why the wavelet-based approach can estimate the software reliability measures with less computational cost than the other nonparametric approaches, and with better goodness-of-fit accuracy than the conventional parametric estimation methods.

In the above studies, the model estimation is done by only point estimation methods. It is well known that point estimation provides only the expectation of certain reliability measure as the output. From a statistical standpoint, the point estimate does not take the uncertainty of the corresponding estimator into consideration. However, it is worth mentioning that the software reliability assessment should be always considering the uncertainty, because the upper and the lower bounds of the target estimator are much more preferable rather than the point estimate itself for practitioners. This motivates us to consider the interval estimation to deal with the uncertainty in software reliability modeling. Kuo and Yang [15], Yin and Trivedi [37], Okamura et al. [19] developed the general approaches for NHPP-based SRMs from the Bayesian statistical points of view. On the other hand, apart from the Bayesian approaches, Yamada and Osaki [35], Joe [12], van Pul [25], Zhao and Xie [38] gave the interval estimates of model parameters in some specific NHPP-based SRMs. These studies are based on asymptotic approximations. Moreover, non-Bayesian interval estimation without asymptotic approximations can be achieved by means of bootstrapping [6]–[8]. The bootstrapping is a combination of data resampling and replication of estimation, and enables us to estimate the probability distribution of arbitrary estimators under interest, if they exist. van Pul [26] applied the statistical bootstrapping technique to the well-known Jelinski and Moranda SRM [17], [18], [22]. Kaneishi and Dohi [13] estimated 95% confidence intervals of software intensity function as well as other software reliability measures of a parametric NHPP-based SRM, using three bootstrap methods.

Our concern in this paper is the confidence interval estimation for nonparametric NHPP-based SRMs, of which the software intensity function is estimated by the wavelet-based approach. There are mainly three advantages of our proposal: (i) It is an interval estimation method that provides the practitioners with a range of the interested software reliability measure. (ii) It is a nonparametric estimation without assumptions under any software debugging sce-
3. Wavelet Shrinkage Estimation for NHPP-Based Software Reliability Models

The applicability and affectivity of the wavelet shrinkage estimation (WSE) in software reliability assessment have been demonstrated in [29] through numerical examples using real software-fault count data. The method is explained here briefly. Consider the problem of estimating the software intensity function \( \lambda_i \) from software-fault count data \( y_i (i = 1, 2, \ldots, n) \) with length \( n \). The basic idea of WSE is to remove the noise included in the observations to get a noise-free estimate of the software intensity function, say \( \hat{\lambda}_i \) \( (i = 1, 2, \ldots, n) \). The noise removal using Haar wavelets is performed through the following three steps: (i) expanding \( y_i \) to obtain the empirical wavelet coefficients, (ii) removing the noise included in the empirical wavelet coefficients using thresholding method, and (iii) making use of the denoised coefficients to calculate \( \hat{\lambda}_i \).

First, expand the observations by

\[
y_i = \sum_{k=0}^{2^{J_0}-1} c_{j_0,k} \phi_{j_0,k}(i) + \sum_{j=j_0}^{\infty} \sum_{k=0}^{2^{j}-1} d_{j,k} \psi_{j,k}(i),
\]

where

\[
c_{j_0,k} = \sum_{i=1}^{n} y_i \phi_{j_0,k}(i),
\]

\[
d_{j,k} = \sum_{i=1}^{n} y_i \psi_{j,k}(i),
\]

are called the empirical scaling coefficients and the empirical wavelet coefficients, respectively, for any primary resolution level \( J_0 \) \( (\geq 0) \). Here, \( \phi_{j,k}(i) \) and \( \psi_{j,k}(i) \) are the so-called Haar father wavelet and Haar mother wavelet, respectively. Due to the implementability, it is reasonable to set an upper limit instead of \( \infty \) for the resolution level \( j \). In other words, the highest resolution level must be finite in practice. We use \( J \) to denote the highest resolution level in this paper. That is, the range of \( j \) in the second term of Eq. (2) is \( j \in [J_0, J] \). Generally, \( J_0 \) is set to be 1, and \( J \) is selected from the range \( j/2^j \leq n \). The mapping from function \( y_i \) to coefficients \( (c_{j_0,k}, d_{j,k}) \) is called the Haar wavelet transform.

Second, the noise involved in the empirical wavelet coefficients \( d_{j,k} \) should be removed. Donoho and Johnstone [2]–[4] and Donoho et al. [5] proposed non-linear wavelet estimators of \( y_i \) based on a reconstruction from judicious selections of the empirical wavelet coefficients, and suggested the extraction of the significant wavelet coefficients by thresholding, where \( d_{j,k} \) are set to 0 if their absolute value is below a certain threshold level. Up to now, although a broad class of thresholding schemes is available in the literature, the common choices of thresholding method are the hard thresholding

\[
\phi_T(u) = u \mathbb{1}_{|u| > \tau},
\]

and the soft thresholding

\[
\phi_S(u) = \text{sgn}(u)(|u| - \tau),
\]

for a fixed threshold level \( \tau \) \( (> 0) \), where \( \mathbb{1}_A \) is the indicator function of an event \( A \), \( \text{sgn}(u) \) is the sign function of \( u \) and \( (u)_+ = \max(0, u) \). There are many methods to determine the threshold level \( \tau \). Kolaczyk [14] developed an appropriate threshold level for Poisson data. They called it level-dependent threshold, which is of the following form:

\[
\tau_j = 2^{-\frac{J_0-j-1}{J_0}} \left\{ 2 \log(2^j) + \sqrt{4 \log^2(2^j) + 8 \lambda_0 \log(2^j) 2^{(J-j)}} \right\}
\]

where \( \lambda_0 \) is the sample mean.

As a result of thresholding, the denoised empirical wavelet coefficients, say \( \hat{d}_{j,k} \), are obtained. Finally, substituting \( (c_{j_0,k}, \hat{d}_{j,k}) \) instead of \( (c_{j_0,k}, d_{j,k}) \) into Eq. (2), we have the estimate \( \hat{\lambda}_i \) \( (i = 1, 2, \ldots, n) \). This procedure is called reconstruction and the corresponding algorithm is called the inverse Haar wavelet transform. Hereafter, we use \( \text{H}(\cdot, \text{ldt}) \) to denote the WSE using a certain thresholding method and the level-dependent threshold (ldt).

In addition, it is known that the estimate obtained from the Haar-wavelet-based denoising procedure tends to have a ‘staircase’-like appearance. This problem can be solved by TIPSH Algorithm [14]. Define a circulant shift operator \( S_h \) by \( (S_h y)_k = y_{(k+h) \mod n} \), where \( y = (y_1, y_2, \ldots, y_n) \). This operator is clearly invertible, i.e., \( S_{-h} = (S_h)^{-1} \). We can obtain the estimate \( \hat{\lambda}_i \) by calculating the fully translation-invariant estimates of \( y \), which is given by

\[
\hat{T}(y; (S_h)_{h \in H}) = \text{Average}_{h \in H} S_{-h}(T(S_h y)),
\]

where \( T(y) \) presents the output of \( \text{H}(\cdot, \text{ldt}) \) with the input \( y \), and \( H = \{0, 1, \ldots, n-1\} \). To distinguish, we use \( \text{HTI}(\cdot, \text{ldt}) \) to denote the \( \text{H}(\cdot, \text{ldt}) \) combined with translation-invariant denoising approach. In this way, the WSE (\( \text{H}(\cdot, \text{ldt}) \) or \( \text{HTI}(\cdot, \text{ldt}) \) provides the point estimate of the software intensity function, from which the point estimate of the mean value function can be obtained by \( \Lambda_i = \sum_{k=0}^{J} \hat{\lambda}_k \) \( (i = 1, 2, \ldots, n) \), where \( \lambda_0 \) = 0.

4. Nonparametric Interval Estimation Method

4.1 Bootstrapping

The bootstrap (BS) method [6]–[8] is a combination of data re-sampling and replication of estimation. Generally, there are two types of BS methods. One is the resampling-based BS method, and the other one is the simulation-based BS method. The resampling-based BS method generates multiple sample data directly from one set of data with replacement. Because the resampling-based BS method is implicitly based on the conditional distribution of inter-failure times of an NHPP, it can be applied to only the software-fault detection time data. In fact, van Pul [26] and Kaneishi and Dohi [13] treated with only the software-fault detection time data. On the other hand, the simulation-based BS method works as long as an estimate of software intensity function is in hand, no matter which type of software.
fault data is used to obtain the estimate. In other words, the simulation-based BS method starts with an estimate that is obtained by applying some kind of estimation method to the observed data, and generates multiple sample data using the estimate. Lewis and Shedlar [16] derived the following theorem:

**THEOREM:** Consider an NHPP \{N(t), t ≥ 0\} with intensity function \( \lambda(t) \). Let \( t_1, t_2, \ldots, t_r \) be the points of the process in the time interval \((0, t_f]\). Suppose that certain function \( \lambda'(t) \) satisfies \( \lambda'(t) ≤ \lambda(t) \) for \( 0 ≤ t ≤ t_f \). For \( j = 1, 2, \ldots, e' \), delete the point \( t_j \) with probability \( 1 - \lambda'(t_j)/\lambda(t_j) \). Then the remaining points form an NHPP \{N*(t), t ≥ 0\} with intensity function \( \lambda'(t) \) in the time interval \((0, t_f]\).

Based on this theorem, they proposed the *thinning* algorithm to generate the arrival times of an NHPP.

**Thinning Algorithm:**
1. Generate points (arrival times) in the NHPP \{N(t), t ≥ 0\} with intensity function \( \lambda(t) \) in the fixed time interval \((0, t_f]\). Denote the ordered points by \( t_1, t_2, \ldots, t_r \). If the number of points generated is such that \( e' = 0 \), exit; there are no points in the generated NHPP \{N*(t), t ≥ 0\} with intensity function \( \lambda'(t) \). Else, set \( j = 1, r = 0 \).
2. Generate \( U_j \), uniformly distributed between 0 and 1. If \( U_j ≤ \lambda'(t_j)/\lambda(t_j) \), accept \( t_j \); set \( r = r + 1 \), \( t'_r = t_j \). Else, reject \( t_j \).
3. Set \( j = j + 1 \). If \( j ≤ e' \), go to 2. Else, go to 4.
4. Set \( n' = r \). Return points \( t'_1, t'_2, \ldots, t'_{n'} \) as generated arrival times of \{N*(t), t ≥ 0\} with \( \lambda'(t) \).

### 4.2 BS method for D-NHPP-Based SRMs

Since the wavelet shrinkage (WS) estimates of the reliability measures such as the software intensity function and the mean value function are based on only one sample data, they do not take the uncertainty of the corresponding WS estimator into consideration. To account for uncertainty, and to meet the need of software reliability analysis with software-fault count data, we employ the simulation-based BS method using thinning algorithm to replicate multiple sets of software-fault count data from the observed data.

In this section, as the main part of the interval estimation method for wavelet-based software reliability assessment, we propose the BS method for D-NHPP-based SRMs. To this end, firstly, the method of thinning in Sect. 4.1 need to be extended. It should be noted that the thinning algorithm is not applicable to simulate NHPP with discrete intensity function. In the initial setting of thinning algorithm, the ordered points \( t_1, t_2, \ldots, t_r \) of original NHPP are continuous values. But when one wants to simulate NHPP with a discrete intensity function \( \lambda'(t), t = 1, 2, \ldots \), the comparison \( U_j ≤ \lambda'(t_j)/\lambda(t_j) \) is not implementable because \( \lambda'(t_j) \) is undefined if \( t_j \) is not an integer. In this paper, we complement the discontinuous points in discrete intensity function with the linear interpolation, which is known as the first-order spline interpolation. More specifically, when generating an NHPP with a discrete intensity function \( \lambda'(t) \), we redefine \( \lambda'(t) \) before the second step of the thinning algorithm as

\[
\lambda'(t) = \begin{cases} 
\lambda'(t) & (t \in \mathbb{Z}) \\
\lambda'(\lfloor t \rfloor) & (\text{otherwise})
\end{cases},
\]

where, \( \lfloor t \rfloor = \max\{s \in \mathbb{Z}|s ≤ t\} \).

By employing the above extended thinning algorithm, we take the following steps to replicate \( m \) sets of BS samples from software-fault count data, which are considered as the realizations of a D-NHPP. Although we use WSE to estimate the software intensity function here, the following procedures are not restricting to WSE. For better understanding, we show the outline of the BS method for D-NHPP-based SRMs in Fig. 1.

- **1.** Let \( y = (y_1, y_2, \ldots, y_n) \) and \( \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n) \) denote the software-fault count data and the discrete software intensity function, respectively.
- **2.** Apply WSE to \( y \) to obtain the WS estimate of \( \lambda \), denoted by \( \hat{\lambda} = (\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_n) \).
- **3.** For \( k = 1, 2, \ldots, m \), generate pseudo random arrival time sequences \( \mathbf{t}_k = (t_{k1}^\star, t_{k2}^\star, \ldots, t_{kn}^\star) \) at the \( k \)-th simulation, via the extended thinning algorithm with \( \hat{\lambda} \). Note that the length \( n^*_k \) is different for each time sequence \( t_{k}^\star \), and \( m \) is a sufficiently large integer. The \( m \) sets of arrival time sequences can be considered as \( m \) sets of software-fault detection time data.
- **4.** For \( k = 1, 2, \ldots, m \), generate BS samples \( y_k^* = (y_{k1}^\star, y_{k2}^\star, \ldots, y_{kn}^\star) \) from \( t_{k}^\star \), by dividing the time-axis of the arrival time sequences equally.
In this algorithm, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n_i^*$, $k = 1, 2, \ldots, m$, where $n$ is the length of software-fault count data, $n_i^*$ is the length of the $k$-th time sequence, and $m$ is the number of BS samples.

**Notations:**
- $\lambda_i$: the discrete software intensity function on the $i$-th testing date $y_i$;
- the number of faults detected on the $i$-th testing date $\hat{\lambda}_i$;
- the wavelet shrinkage estimate of $\lambda_i$ using $n_i^*$;
- $t_i^*$: the $j$-th time point in the $k$-th time sequence $y_i^*$;
- the number of faults on the $i$-th testing date in the $k$-th BS sample $\hat{\lambda}_{i,j}$;
- the wavelet shrinkage estimate of $\lambda_i$ using $n_i^*$;
- WSE: wavelet shrinkage estimation with input $()$.

**Step 0. Initial setting:**
- For $i = 1, 2, \ldots, n$, Set $\hat{\lambda}_i = 0$.
- For $k = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n_i^*$, Set $\hat{\lambda}_{i,j} = 0$.

**Step 1. Obtain point estimate of $\lambda_i$:** Set $\hat{\lambda}_i = \text{WSE}(y_i)$.

**Step 2. Generate $t_i^*$ by extended thinning algorithm:**
1. Set $\hat{\lambda} = \max(\hat{\lambda}(j))$, where $\hat{\lambda}(j) = \begin{cases} \hat{\lambda}_i & (\text{if } t \in \mathbb{Z}) \\ \lambda_i & (\text{when } i \notin \mathbb{Z}) \end{cases}$.
2. For $k = 1, 2, \ldots, m$, Do:
   2.1: Set $\hat{\lambda}_j = 0$; $\hat{j} = 0$; $j = 1$.
   2.2: Generate $U \sim U(0, 1)$; Set $E = \ln(U/\hat{\lambda})$; $\hat{\lambda} = \hat{\lambda}^* = E$.
   2.3: If $E < \hat{\lambda}^*$ then $\hat{\lambda}^* = \hat{\lambda}^*$, else $j = j + 1$.
   2.4: If $\hat{\lambda}^* \leq \hat{\lambda}$ then $j = j + 1$; go to step 2.2, else go to 2.

**Step 3. Obtain $n$ sets of point estimates of $\lambda_i$:**
1. For $k = 1, 2, \ldots, m$, $i = 1, 2, \ldots, n$, Set $y_{i,j}^* = n_{i,j}^*$ as such as $1 < n_{i,j}^* < i$.
2. For $k = 1, 2, \ldots, m$, Set $y_{i,j}^* = \text{WSE}(y_{i,j}^*)$.

**Step 4. Estimate the empirical distribution of $\lambda_i^*$:**
1. For $i = 1, 2, \ldots, n$, Do:
   1.1: Sort $\hat{\lambda}_{i,j}$ as $\hat{\lambda}_{i,0} \leq \hat{\lambda}_{i,1} \leq \hat{\lambda}_{i,2} \leq \cdots \leq \hat{\lambda}_{i,n_{i,j}^*}$.
   1.2: Set $G_{i,n_{i,j}^*} = \begin{cases} k/m & \text{for } \hat{\lambda}_{i,j} \leq \hat{\lambda}_{i,j+1} \\ 1 & \text{for } \hat{\lambda}_{i,n_{i,j}^*} \leq \hat{\lambda} \end{cases}$.

**Step 5. Obtain the two-sided 100$p$% confidence interval of $\lambda_i^*$:**
1. Obtain the lower bound $\hat{\lambda}_{i,(k)}$, where $k_i$ corresponds to $100(1 - p)$%-quantile satisfying $G_{i,n_{i,j}^*}(\hat{\lambda}_{i,(k_i)}) = 100(1 - p)$.
2. Obtain the upper bound $\hat{\lambda}_{i,(m)}$, where $k_i$ corresponds to $100p$%-quantile satisfying $G_{i,n_{i,j}^*}(\hat{\lambda}_{i,(m)}) = 100p$.

Fig. 2 The pseudo code of confidence interval estimation for wavelet-based software reliability assessment.

5. For $k = 1, 2, \ldots, m$, apply WSE to $y_{i,k}^*$ to obtain the WS estimates of $\lambda_i$, denoted by $\hat{\lambda}_i^* = (\hat{\lambda}_{i,1}^*, \hat{\lambda}_{i,2}^*, \ldots, \hat{\lambda}_{i,m}^*)$.

### 4.3 Confidence Interval Estimation of Software Reliability Measures

Now we are at the point of estimating the confidence interval of the WS estimator of the software intensity function $\lambda_i$ with $m$ sets of WS estimates $\hat{\lambda}_i^* = (\hat{\lambda}_{i,1}^*, \hat{\lambda}_{i,2}^*, \ldots, \hat{\lambda}_{i,m}^*)$ in hand. Define the order statistics of the software intensity function $0 = \hat{\lambda}_{i,(0)}^* \leq \hat{\lambda}_{i,(1)}^* \leq \hat{\lambda}_{i,(2)}^* \leq \cdots \leq \hat{\lambda}_{i,(m)}^*$ at time $i$, and regard them as the complete sample from random variable $\lambda_i^*$, which represents the number of software faults detected at time $i$. If there exists the cumulative distribution function (c.d.f.) of $\lambda_i^*$, the empirical c.d.f. $G_{i,n_{i,j}^*}(\lambda)$ corresponding to the sample estimates $\hat{\lambda}_{i,(k)}^*$ ($k = 0, 1, 2, \ldots, m$) is given by

$$G_{i,n_{i,j}^*}(\lambda) = \begin{cases} k/m & \text{for } \hat{\lambda}_{i,(k)}^* \leq \lambda \leq \hat{\lambda}_{i,(k+1)}^* \\ 1 & \text{for } \hat{\lambda}_{i,(m)}^* \leq \lambda \end{cases}$$

(10)

It is well known that the above empirical c.d.f. approaches to the real (but unknown) c.d.f. of $\lambda_i^*$ as $m \to \infty$ and $n \to \infty$, and is strongly consistent. From the empirical c.d.f. of $\lambda_i^*$, we can obtain not only the mean $E[\lambda_i^*]$ and its higher moments such as $E[(\lambda_i^*)^2]$, $E[(\lambda_i^*)^3]$, $E[(\lambda_i^*)^4]$, etc., but also the confidence interval with significant level $p$. The confidence interval of $\lambda_i^*$ is defined as the quantile of Eq. (10), so that the two-sided 100$p$% confidence interval is given by $[\hat{\lambda}_{i,(k)}, \hat{\lambda}_{i,(m)}]$, where $k_L$ and $k_U$ are indexes corresponding to $100(1 - p)$% and $100p$%-quantiles satisfying $G_{i,n_{i,j}^*}(\hat{\lambda}_{i,(k)_L}) = 100(1 - p)$ and $G_{i,n_{i,j}^*}(\hat{\lambda}_{i,(k)_U}) = 100p$. The similar approach can be taken to the other reliability measures such as the mean value function. To complete the discussion, we give the pseudo code of the interval estimation method for wavelet-based SRMs in Fig. 2.

### 5. Numerical Analysis

#### 5.1 Data Set and Experimental Setting

We use a real software project data set cited in Ref. [17] where it is named as J1. This is a set of software-fault count data (group data), where the number of software faults detected on each testing date is recorded. The final testing date and the total cumulative number of detected faults of this data are 62 and 133, respectively. As we apply the WSE with two thresholding techniques (hard thresholding (h) v.s. soft thresholding (s)) using level-dependent threshold (ldt), the results of a total number of four WSEs are shown in this paper. More specifically, the four methods are H(h, ldt), H(s, ldt), HTI(h, ldt), and HTI(s, ldt). Based on a preliminary empirical study [31], we have found that resolution level $J = 3$ provided better goodness-of-fit performance than $J = 4$ or $J = 5$. For brevity, we restrict the following discussions to the WSEs with resolution level $J = 3$.

We generate $m = 10000$ BS samples based on the BS method for D-NHPP-based SRMs. Throughout this paper, we fix the significance level as 5%. That is, the 95% confidence intervals of both software intensity function and the mean value function are derived. In addition, the mean, median, variance, skewness, kurtosis of the WS estimators are also presented in Table 1 and Table 2, at four observation points (25%, 50%, 75% and 100%) of the whole data. Note that the evaluations are executed at all observation points, but we show only four representative points to give a global insight of the evaluation results.

#### 5.2 Results and Discussions

##### 5.2.1 On the Mean and the Median

It is observed from Table 1 and Table 2 that the mean of the
Table 1  Estimation Results with Real Software Fault Count Data (software intensity function).

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>median</th>
<th>variance</th>
<th>skewness</th>
<th>kurtosis</th>
<th>confidence interval</th>
<th>point estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25%</td>
<td>2.209</td>
<td>2.078</td>
<td>1.406</td>
<td>0.691</td>
<td>[0.327, 4.892]</td>
<td>2.703</td>
</tr>
<tr>
<td>H(h, ldt)</td>
<td>50%</td>
<td>2.664</td>
<td>2.563</td>
<td>1.290</td>
<td>0.544</td>
<td>[0.698, 5.189]</td>
<td>2.438</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>2.365</td>
<td>2.203</td>
<td>1.363</td>
<td>0.724</td>
<td>[0.537, 5.017]</td>
<td>0.625</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>1.443</td>
<td>1.375</td>
<td>0.653</td>
<td>0.958</td>
<td>[0.127, 3.377]</td>
<td>1.750</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>2.313</td>
<td>2.266</td>
<td>0.555</td>
<td>0.645</td>
<td>[1.029, 3.984]</td>
<td>2.703</td>
</tr>
<tr>
<td>H(s, ldt)</td>
<td>50%</td>
<td>2.416</td>
<td>2.359</td>
<td>0.530</td>
<td>0.805</td>
<td>[1.178, 4.070]</td>
<td>2.358</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>2.345</td>
<td>2.297</td>
<td>0.537</td>
<td>0.632</td>
<td>[1.079, 3.942]</td>
<td>2.160</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>1.651</td>
<td>1.618</td>
<td>0.541</td>
<td>0.930</td>
<td>[0.439, 3.364]</td>
<td>0.973</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>2.209</td>
<td>2.078</td>
<td>1.406</td>
<td>0.691</td>
<td>[0.327, 4.892]</td>
<td>2.703</td>
</tr>
<tr>
<td>HTI(h, ldt)</td>
<td>50%</td>
<td>2.664</td>
<td>2.563</td>
<td>1.290</td>
<td>0.544</td>
<td>[0.698, 5.189]</td>
<td>2.438</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>2.365</td>
<td>2.203</td>
<td>1.363</td>
<td>0.724</td>
<td>[0.537, 5.017]</td>
<td>0.625</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>1.443</td>
<td>1.375</td>
<td>0.653</td>
<td>0.958</td>
<td>[0.127, 3.377]</td>
<td>1.750</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>2.313</td>
<td>2.266</td>
<td>0.555</td>
<td>0.645</td>
<td>[1.029, 3.984]</td>
<td>2.703</td>
</tr>
<tr>
<td>HTI(s, ldt)</td>
<td>50%</td>
<td>2.416</td>
<td>2.359</td>
<td>0.530</td>
<td>0.805</td>
<td>[1.178, 4.070]</td>
<td>2.358</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>2.345</td>
<td>2.297</td>
<td>0.537</td>
<td>0.632</td>
<td>[1.079, 3.942]</td>
<td>2.160</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>1.651</td>
<td>1.618</td>
<td>0.541</td>
<td>0.930</td>
<td>[0.439, 3.364]</td>
<td>0.973</td>
</tr>
</tbody>
</table>

Table 2  Estimation Results with Real Software Fault Count Data (mean value function).

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>median</th>
<th>variance</th>
<th>skewness</th>
<th>kurtosis</th>
<th>confidence interval</th>
<th>point estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25%</td>
<td>34.546</td>
<td>34.338</td>
<td>43.622</td>
<td>0.176</td>
<td>[22.079, 47.953]</td>
<td>39.692</td>
</tr>
<tr>
<td>H(h, ldt)</td>
<td>50%</td>
<td>69.113</td>
<td>68.989</td>
<td>95.097</td>
<td>0.092</td>
<td>[50.624, 88.657]</td>
<td>70.926</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>105.698</td>
<td>105.605</td>
<td>148.972</td>
<td>0.067</td>
<td>[82.067, 129.880]</td>
<td>103.152</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>138.079</td>
<td>137.989</td>
<td>202.285</td>
<td>0.059</td>
<td>[110.178, 166.656]</td>
<td>134.954</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>34.546</td>
<td>34.338</td>
<td>43.622</td>
<td>0.176</td>
<td>[22.079, 47.953]</td>
<td>39.692</td>
</tr>
<tr>
<td>H(s, ldt)</td>
<td>50%</td>
<td>67.957</td>
<td>67.925</td>
<td>95.097</td>
<td>0.092</td>
<td>[50.624, 88.657]</td>
<td>70.926</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>102.732</td>
<td>102.561</td>
<td>148.657</td>
<td>0.104</td>
<td>[79.416, 127.543]</td>
<td>104.008</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>135.834</td>
<td>135.500</td>
<td>212.086</td>
<td>0.105</td>
<td>[108.099, 165.355]</td>
<td>133.543</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>34.446</td>
<td>34.320</td>
<td>47.458</td>
<td>0.160</td>
<td>[21.268, 48.278]</td>
<td>39.334</td>
</tr>
<tr>
<td>HTI(h, ldt)</td>
<td>50%</td>
<td>67.957</td>
<td>67.925</td>
<td>95.097</td>
<td>0.092</td>
<td>[50.624, 88.657]</td>
<td>70.926</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>102.732</td>
<td>102.561</td>
<td>148.657</td>
<td>0.104</td>
<td>[79.416, 127.543]</td>
<td>104.008</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>135.834</td>
<td>135.500</td>
<td>212.086</td>
<td>0.105</td>
<td>[108.099, 165.355]</td>
<td>133.543</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>34.446</td>
<td>34.320</td>
<td>47.458</td>
<td>0.160</td>
<td>[21.268, 48.278]</td>
<td>39.334</td>
</tr>
<tr>
<td>HTI(s, ldt)</td>
<td>50%</td>
<td>67.957</td>
<td>67.925</td>
<td>95.097</td>
<td>0.092</td>
<td>[50.624, 88.657]</td>
<td>70.926</td>
</tr>
<tr>
<td></td>
<td>75%</td>
<td>102.732</td>
<td>102.561</td>
<td>148.657</td>
<td>0.104</td>
<td>[79.416, 127.543]</td>
<td>104.008</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>135.834</td>
<td>135.500</td>
<td>212.086</td>
<td>0.105</td>
<td>[108.099, 165.355]</td>
<td>133.543</td>
</tr>
</tbody>
</table>

For comparison, we show the point estimates of WSEs at corresponding observation points in the last columns of Table 1 and Table 2. Comparing with the WS estimates of the mean value function, we can see that the BS estimates (median) based on WSEs with hard thresholding are smaller at observation points 25% and 50%, while are larger at observation points 75% and 100%. In case of WSEs with soft thresholding, the BS estimates are smaller than the WS estimates also at observation point 75%. A similar trend can be found in case of software intensity function. Therefore, it may be said that the BS estimates tend to be rather underestimated than the WS estimates in early observation points. In fact, the MSE between the WS estimates and the real data is 0.197749, which is much smaller than that of the BS estimates.
As shown in Fig. 3, the WS estimates are very close to the real data. However, this result does not mean that the BS estimates are inferior to the WS estimates, but implies that the WS estimates may be over-fitted to the real data. From a statistical viewpoint, the BS estimates are rather reasonable.

5.2.2 On the Variance and the Confidence Interval

It is found that HTI(s, ldt) shows the smallest variance at observation points 25% and 50% in Table 1, while HTI(h, ldt) provides the smallest variance at three observation points (50%, 75% and 100%) in Table 2. These two WSEs also provide relatively tight 95% confidence interval. More precisely, the WSEs that give the narrowest confidence interval at each observation point are as shown in Table 3. Because a narrow confidence interval indicates that the population value is probably quite close to the sample estimate, and a wide confidence interval indicates that the population value may be quite far from the sample estimate, we suggest that the WSEs with hard thresholding are the better ones for the mean value function, while the WSEs with soft thresholding should be used when estimating the software intensity function. Figure 4 depicts the behavior of both the WS estimates and their confidence intervals. The bar graph and the solid line indicate the observed data and the WS estimates, respectively, while the gray-shaded boxes show the 95% confidence intervals at observation points 25%, 50%, 75% and 100% of the whole data. For example, from (ii) of Fig. 4, we can know the upper and the lower bounds of expected cumulative number of software faults.

For the purpose of comparison with existing parametric NHPP-based SRMs, we apply the simulation-based BS method to an NHPP-based SRM with geometric distribution. The maximum likelihood (ML) estimation is used for estimating the unknown model parameters. Figure 5 depicts the behavior of the confidence intervals of both WS estimators and ML estimators. It can be confirmed from this figure that, i) the real software-fault count data always inside both confidence intervals, and ii) the confidence interval of WS estimators is narrower than that of ML estimators. The point i) ensures that our interval estimation method is reliable, and the point ii) suggests that our method is superior to the existing ones. Moreover, the confidence interval of the mean value function enables us to obtain a comprehensive view in decision making.

Furthermore, it is worth mentioning that HTI(h, ldt) outperforms the other WSEs in terms of goodness-of-fit performance, which was confirmed in our preliminary empirical study [29]. Therefore, from the numerical examples on
Table 4  Approximated Confidence Intervals by Normal Distribution. (mean value function)

<table>
<thead>
<tr>
<th>J = 3</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(h, ldt)</td>
<td>25% [21.601, 47.491]</td>
</tr>
<tr>
<td></td>
<td>50% [50.000, 88.226]</td>
</tr>
<tr>
<td></td>
<td>75% [81.776, 129.620]</td>
</tr>
<tr>
<td></td>
<td>100% [110.203, 165.955]</td>
</tr>
<tr>
<td>H(s, ldt)</td>
<td>25% [20.944, 47.948]</td>
</tr>
<tr>
<td></td>
<td>50% [48.633, 87.281]</td>
</tr>
<tr>
<td></td>
<td>75% [78.835, 126.623]</td>
</tr>
<tr>
<td></td>
<td>100% [107.291, 163.377]</td>
</tr>
<tr>
<td>HTI(h, ldt)</td>
<td>25% [21.734, 48.368]</td>
</tr>
<tr>
<td></td>
<td>50% [49.037, 86.623]</td>
</tr>
<tr>
<td></td>
<td>75% [79.789, 126.797]</td>
</tr>
<tr>
<td></td>
<td>100% [107.504, 161.950]</td>
</tr>
<tr>
<td>HTI(s, ldt)</td>
<td>25% [21.197, 48.077]</td>
</tr>
<tr>
<td></td>
<td>50% [48.303, 86.741]</td>
</tr>
<tr>
<td></td>
<td>75% [78.298, 126.196]</td>
</tr>
<tr>
<td></td>
<td>100% [106.945, 163.197]</td>
</tr>
</tbody>
</table>

Table 5  Percentage of Differences. (mean value function)

<table>
<thead>
<tr>
<th>J = 3</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(h, ldt)</td>
<td>25% (2.17, 0.96)</td>
</tr>
<tr>
<td></td>
<td>50% (1.23, 0.49)</td>
</tr>
<tr>
<td></td>
<td>75% (0.36, 0.20)</td>
</tr>
<tr>
<td></td>
<td>100% (-0.02, 0.42)</td>
</tr>
<tr>
<td>H(s, ldt)</td>
<td>25% (1.52, 0.68)</td>
</tr>
<tr>
<td></td>
<td>50% (0.94, 0.64)</td>
</tr>
<tr>
<td></td>
<td>75% (0.73, 0.47)</td>
</tr>
<tr>
<td></td>
<td>100% (0.71, 0.63)</td>
</tr>
<tr>
<td>HTI(h, ldt)</td>
<td>25% (2.27, 0.87)</td>
</tr>
<tr>
<td></td>
<td>50% (1.86, 1.06)</td>
</tr>
<tr>
<td></td>
<td>75% (1.10, 0.59)</td>
</tr>
<tr>
<td></td>
<td>100% (0.63, 0.14)</td>
</tr>
<tr>
<td>HTI(s, ldt)</td>
<td>25% (0.71, 0.63)</td>
</tr>
<tr>
<td></td>
<td>50% (0.36, 0.20)</td>
</tr>
<tr>
<td></td>
<td>75% (0.94, 0.64)</td>
</tr>
<tr>
<td></td>
<td>100% (0.36, 0.20)</td>
</tr>
</tbody>
</table>

both the point estimation and the interval estimation, it can be concluded that HTI(h, ldt) is the most appreciate wavelet shrinkage-based estimation method among the four WSEs.

5.2.3  On the Skewness and the Kurtosis

Finally, looking at Table 1 and Table 2, it is found to be right-skewed empirical c.d.f.s for the WS estimates of both the software intensity function and the mean value function. While the kurtosis shown in Table 1 are relatively far from 3, it is evident from Table 2 that the resulting empirical c.d.f.s of the WS estimator of the mean value function are considerably near to a normal distribution. In this case, an asymptotic normal approximation can be applied to the estimation of mean value function. We show the confidence intervals of normal distribution in Table 4. The mean and the variance are set to be the same as those of the empirical c.d.f.s of corresponding WS estimator. For instance, the first row [21.601, 47.491] is the result from the normal approximated confidence intervals of the mean value function. We predict the confidence intervals of the mean value function by introducing the idea of one-stage look-ahead prediction [30].

Fig. 6  Probability distributions of the mean value function (HTI(h, ldt)).

Table 5. It can be seen that, except for observation point 25%, the differences are smaller than 1% in most cases. As an example, we plot the empirical c.d.f.s based on HTI(h, ldt) at four observation points (25%, 50%, 75% and 100%) in Fig. 6. In addition, the corresponding normal distributions with the same mean and variance are plotted in solid lines. More specifically, the normal distributions in (i) ~ (iv) of Fig. 6 are N(35.051, 46.167), N(67.830, 91.943), N(103.293, 143.812), and N(134.727, 192.915). It can be seen from these figures that, the solid lines and the gray-shaded bar graphs overlap considerably. This implies that the normal approximation is acceptable for estimating the mean value function.

5.3  Software Reliability Assessment

Our next concern is to develop a prediction method for the proposed interval estimation method. We predict the confidence intervals of the software reliability measures by introducing the idea of one-stage look-ahead prediction [30]. More specifically, the prediction is executed at the 2-th testing date. That is, we regard the data observed from t = 2 to n as unknown data, and predict the 95% confidence intervals of the WS estimators based on the data observed from t = 1 to t = 2. Here, we show the behavior of the confidence intervals of the mean value function in Fig. 7. Because we set the resolution level J to 3, the confidence intervals and the median of the WS estimator start from
software reliability sequentially. Figure 8 plots the sequential reliability measures, one can predict the quantitative software reliability. This weakness motivates us to improve our method in future work.

Using the predicted confidence interval of the software reliability measures, one can predict the quantitative software reliability sequentially. Figure 8 plots the sequential estimation of the software reliability function, where the software reliability is the probability that no faults are experienced at time interval \([t, t+1]\), and is given by

\[
R(1 | t) = \exp \{-[\Lambda_{t+1} - \Lambda_t]\} = \exp \{-\lambda_{t+1}\}, \quad (t = 2^j, 2^j + 1, \ldots)
\] (13)

Note that the software reliability function shown in Fig. 8 is sequential estimates, so it does not have monotonically decreasing property. However, it is found that the upper bound of the software reliability seems to be globally-decreasing. In addition, the width between the upper and the lower bounds tends to become narrower over time. This result will be very helpful in practice, even though the behavior of the predictive software reliability is quite chaos. For example, this figure tells us that, even in the worst case, the software reliability is still larger than 0.1 on the 25-th unit time after release. However, even in the best case, the software reliability will drop down to 0.18 on the 25-th unit time. Then the project leader can decide to execute additional software test if he or she thinks the software reliability is too low. In this case, a higher reliability will be gained, but the shipping time may be delayed, and more development resources will be required. On the other hand, the life cycle of the current software product may be no longer than 10 unit time (for example, 10 months), and the budget may be rather tight. In this case, the project leader can choose to release the software to meet the budget and the delivery deadline. In this way, the proposed interval estimation method can provide the practitioners a useful measure in flexible decision making.

6. Conclusion

In this paper, we have developed the interval estimation method for the wavelet-based software reliability assessment. The simulation-based bootstrap method has been utilized to derive the interval estimates for both the software intensity function and the mean value function, where the underlying stochastic process was described by the well-known NHPP-based SRMs. It should be noted that this was the first paper to estimate the two-sided confidence intervals of wavelet estimate using software-fault count data. We have estimated the empirical distributions of the nonparametric wavelet estimators, and have obtained the higher moments of the above estimators, such as variance, skewness and kurtosis, and the 95% confidence intervals.

The theoretical contribution of this paper is being succeeded in extending the conventional simulation-based BS method to be applicable to discrete NHPP-based SRMs. Equation (9) is the key idea of this extension. As the first attempt, we applied the linear interpolation, which is the first-order spline interpolation. In the future, we will use high-order spline interpolation to reduce the approximation error caused by the interpolation method. The first practical contribution of this paper is making it possible to derive confidence intervals of software reliability measures from software-fault count data, based on nonparametric NHPP-based SRMs. In the testing phase of software development, the collection of software-fault count data is rather easier than that of software-fault detection time data. Therefore, it is preferable to develop interval estimation method that treats with software-fault count data. Furthermore, this paper provides the software development team with a new methodology. The upper and the lower bounds of predictive software reliability have been calculated. From the result of the numerical study, it has been found that our method is very helpful in grasping an appropriate perception of the present situation of the product quality. In the future, we will conduct a large-scale experiment with more data sets to establish the credibility and usefulness of the proposed method.

Although we have only focused on the software-fault count data, the confidence interval estimation method for analyzing the software-fault detection time data can also be considered in a similar fashion. In the future, we will propose more general interval estimation framework for a wider class of wavelet-based software reliability analysis.
References


Tadashi Dohi received the B.Sc. (Engineering), M.Sc. (Engineering) and Ph.D. (Engineering) from Hiroshima University, Japan, in 1989, 1991 and 1995, respectively. Since 2002, he has been working as a Full Professor in the Department of Information Engineering, Graduate School of Engineering, Hiroshima University. He is a Regular Member of ORSJ, JSIAM, IEICE, REAJ and IEEE.