Discrete Time-Space Stochastic Mathematical Modelling for Quantitative Description of Software Imperfect Fault-Debugging with Change-Point

'Mohd Taib' Shatnawi[1]*
Al-Huson University College
Al-Balqa' Applied University
Irbid 21510, Jordan

Omar Shatnawi[2]
Computer Science Department
Al al-Bayt University
Mafraq 25113, Jordan

***Abstract*—Statistics and stochastic-process theories, along with the mathematical modelling and the respective empirical evidence support, describe the software fault-debugging phenomenon. In software-reliability engineering literature, stochastic mathematical models based on the non-homogeneous Poisson process (NHPP) are employed to measure and boost reliability too. Since reliability evolves on account of the running of computer test-run, NHPP type of discrete time-space models, or difference-equation, is superior to their continuous time-space counterparts. The majority of these models assume either a constant, monotonically increasing, or decreasing fault-debugging rate under an imperfect fault-debugging environment. However, in the most debugging scenario, a sudden change may occur to the fault-debugging rate due to an addition to, deletion from, or modification of the source code. Thus, the fault-debugging rate may not always be smooth and is subject to change at some point in time called change-point. Significantly few studies have addressed the problem of change-point in discrete-time modelling approach. The paper examines the combined effects of change-point and imperfect fault-debugging with the learning process on software-reliability growth phenomena based on the NHPP type of discrete time-space modelling approach. The performance of the proposed modelling approach is compared with other existing approaches on an actual software-reliability dataset cited in literature. The findings reveal that incorporating the effect of change-point in software-reliability growth modelling enhances the accuracy of software-reliability assessment because the stochastic characteristics of the software fault-debugging phenomenon alter at the change-point.**

Keywords*—Stochastic mathematical modelling; discrete time-space; non-homogenous poisson process; change-point; imperfect fault-debugging; software-reliability

I. Introduction

Today, computer-based software systems are indispensable, and their successful operation depends mainly on software. A fault can be introduced at any point during the software development life cycle (abbreviated as SDLC) due to the deficiency of human-being. A failure is a consequence of a fault. The software development process's testing phase aims at debugging faults. Software reliability is typically defined as a statistical measure of a software system's ability to operate failure-free. To quantify software reliability, a number of analytical approaches to mathematical modelling the stochastic-behavior of software debugging phenomenon have been proposed. Non-homogeneous Poisson process (abbreviated as NHPP) models are widely used in software-reliability engineering to quantitatively express reliability. They are broadly divided into discrete time-space and continuous time-space groups. However, discrete time-space models which adopt the number of executed computer test-run as a unit of fault-debugging period are more suited than their continuous time-space counterparts. Despite the difficulties in formulating them, many studies have highlighted their usefulness [1]–[4].

Several existing NHPP stochastic software-reliability models are based on constant or monotonically increasing fault-debugging rates under perfect and imperfect fault-debugging environments [1], [3], [5]–[16]. In practice, as the debugging grows, a sudden change may take place to the fault-debugging rate as a result of an addition to, deletion from, or modification of the source code. Thus, the fault-debugging rate may not always be smooth and is subject to change at some point in time called change-point [1], [17]–[23]. Many studies argue that considering the change-point concept in imperfect fault-debugging is expected to enhance the software-reliability assessment accuracy due to the stochastic characteristics of software fault-debugging phenomenon changed at the change-point. Thus, the incorporation of change-point provides a notable enhancement in the reliability assessment. However, most of the endeavors are in continuous time-space. Lately, this area has received little attention and few studies have incorporated the change-point concept in developing discrete time-space software-reliability modelling [1], [24], [25]. However, the study of the change-point problem is still very limited in the discrete-time modelling approach. As a result, the paper studies the effect of incorporating a change-point concept in modelling the imperfect fault-debugging phenomenon.

The rest of the paper is structured as follows: Section II reviews related work and models the change-point problem into an imperfect fault-debugging environment through a discrete time-space NHPP based approach. Data analyses and parameter estimation techniques, model validation, comparison criteria, and software-reliability measures are discussed in Sections III and IV, respectively. The descriptive-performance and predictive-capability, and software-reliability evaluation measures based on the proposed modelling approach are shown in Section V. Finally, Section VI concludes the paper and presents future works.

*Corresponding author.

II. Software-Reliability Modelling: A Discrete Time-Space Approach

In this approach, we model the debugged faults by n computer test-run as a pure-birth counting process $(N_n; n \geq 0)$, [1]–[3] subject to

- No fault debugged at $n = 0$, that is, $N(0) = 0$.
- For any number of computer test-run $n_1, n_2, ..., n_k$ where $(0 < n_1 < n_2 < ... < n_k)$. The k random variables $N(n_1)$, $N(n_2) - N(n_1)$, ..., $N(n_k) - N(n_{k-1})$ are statistically independent.
- For any number of computer test-run n_i and n_j, where $(0 \leq n_i \leq n_j)$, we have

$$Pr[N(n_j) - N(t_i) = k] = \frac{(\lambda_n)^k}{k!} e^{-\lambda_n} \tag{1}$$

here λ be the fault-debugging intensity function.

Accordingly, the NHPP software-reliability model can be formulated

$$Pr[N_n = k] = \frac{(m_n)^k}{k!} e^{-m_n} \tag{2}$$

here N_n be the expected value number of faults whose mean-value function (abbreviated as MVF) is known as m_n.

A. Model Development

In this section, we review some of the well-documnetd NHPP type of discrete time-space models

1) Exponential Model [3], [26]: This model considers that the average number of faults debugged between n^{th} and the $(n+1)^{th}$ computer test-run, is corresponding to the current number of undebugged faults after the execution of the n^{th} computer test-run, meets the resulting difference equation:

$$\lambda_n = \frac{m_{n+1} - m_n}{\delta} = b(a - m_n) \tag{3}$$

here a be a fault-content, and the constant b represents a fault-debugging per undebugged faults per test case.

To solve the difference equation (3), we employ the probability generating function (abbreviated as PGF) computational technique, multiply both sides by w^n, and sum over n from 0 to ∞, we get:

$$\sum_{n=0}^{\infty} w^n m_{n+1} - \sum_{n=0}^{\infty} w^n m_n = \delta b(a \sum_{n=0}^{\infty} w^n - \sum_{n=0}^{\infty} w^n m_n)$$

$$\frac{1}{w} \sum_{n=0}^{\infty} w^{n+1} m_{n+1} - \sum_{n=0}^{\infty} w^n m_n = \delta ab \sum_{n=0}^{\infty} w^n - \delta b \sum_{i=0}^{\infty} w^n m_n$$

$$\sum_{n=0}^{\infty} w^{n+1} m_{n+1} - \sum_{n=0}^{\infty} w^{n+1} m_n = \delta ab \sum_{n=0}^{\infty} w^{n+1} - \delta b \sum_{n=0}^{\infty} w^{n+1} m_n$$

$$\sum_{n=0}^{\infty} w^{n+1} m_{n+1} = \delta ab \sum_{n=0}^{\infty} w^{n+1} - (1 - \delta b) \sum_{n=0}^{\infty} w^{n+1} m_n$$

$$(w^1 m_1 + w^2 m_2 + w^3 m_3 + ...) = \delta ab(w^1 + w^2 + w^3 + ...) + (1 - \delta b)(w^1 m_0 + w^2 m_1 + w^3 m_2 + ...) \tag{4}$$

Comparing the coefficients of like powers of w on both sides in (4) and under the initial-condition $m_{n=0} = 0$, yields

$$\begin{aligned} m_1 &= \delta ab + (1 - \delta b) m_0 = a(1 - (1 - \delta b)^1) \\ m_2 &= \delta ab + (1 - \delta b) m_1 = a(1 - (1 - \delta b)^2) \\ m_3 &= \delta ab + (1 - \delta b) m_2 = a(1 - (1 - \delta b)^3) \end{aligned} \tag{5}$$

The closed-form solution, by mathematical-induction, obtained as

$$m_n = a(1 - (1 - \delta b)^n) \tag{6}$$

Accordingly, the fault-debugging intensity function is

$$\lambda_n = \frac{m_{n+1} - m_n}{\delta} = ab(1 - \delta b)^n \tag{7}$$

The equivalent continuous time-space model [27] corresponding to (6) is

$$m_t = a(1 - e^{-bt}) \tag{8}$$

which derived as a limiting case of discrete time-space model substituting $t = n\delta$, $\lim_{x \to 0}(1 + x)^{1/x} = e$, and taking limit $\delta \to 0$.

2) Delayed S-shaped Model [1], [16]: This model considers that the fault-debugging phenomenon as a two-stage process namely, detection and correction. Accordingly, we have

$$\begin{aligned} \frac{m_{d_{(n+1)}} - m_{d_{(n)}}}{\delta} &= b(a - m_{d_{(n)}}) \\ \frac{m_{c_{(n+1)}} - m_{c_{(n)}}}{\delta} &= b(m_{d_{(n)}} - m_{c_{(n)}}) \end{aligned} \tag{9}$$

Solving the system of difference equation (9), using the PGF computational technique in terms of the initial condition that at $n = 0$, $m_{d_{(n)}} = m_{c_{(n)}} = 0$, we can obtain the closed-form exact solution as

$$m_n \equiv m_{c_{(n)}} = a((1 + \delta bn)(1 - \delta b)^n) \tag{10}$$

An alternate formulation of (9), to obtain it in single stage, is

$$\frac{m_{n+1} - m_n}{\delta} = b_{n+1}(a - m_n)$$

where

$$b_{n+1} = \frac{\delta b^2 n}{1 + \delta bn} \tag{11}$$

Solving (11), using the PGF computational technique in terms of the initial condition that at $n = 0$, $m_n = 0$, we can obtain the closed-form exact solution as

$$m_n = a((1 + \delta bn)(1 - \delta b)^n) \tag{12}$$

It can be noticed that (12) and (11) are same.

Accordingly, the fault-debugging intensity function is

$$\lambda_n = \frac{m_{n+1} - m_n}{\delta} = \frac{\delta ab^2 n}{1 + \delta bn}((1 + \delta bn)(1 - \delta b)^n) \tag{13}$$

The equivalent continuous time-space model [3], [28] corresponding to (10) or (12) is

$$m_t = a(1 - (1 + bt)e^{-bt}) \tag{14}$$

which derived as a limiting case of discrete time-space model substituting $t = n\delta$, $\lim_{x \to 0}(1 + x)^{1/x} = e$, and taking limit $\delta \to 0$.

3) Inflection S-shaped Model [1], [29], [30]: The primary assumption of the models mentioned above is that the fault-debugging rate depends linearly upon the number of undebugged faults. In practice, it has been observed that as debugging progress, the fault debugging rate has three possible trends: decreasing, constant, and increasing. To analyze these trends, we interpret the fault debugging rate as a function of the number of executed computer test-run. Accordingly, we have the following difference equation:

$$\frac{m_{n+1} - m_n}{\delta} = b_{n+1}(a - m_n)$$

where

$$b_{n+1} = b_i + (b_f - b_i)\frac{m_{n+1}}{a} \tag{15}$$

Solving (15), using the PGF computational technique in terms of the initial condition that at $n = 0$, $m_n = 0$, we can obtain the closed-form exact solution as

$$m_n = a\frac{1 - (1 - \delta b_f)^n}{1 + \frac{b_f - b_i}{b_i}(1 - \delta b_f)^n} \tag{16}$$

The growth curve's shape is determined by the parameters b_i and b_f and can be either exponential or S-shaped. However, (16) can be rewritten to define a constant fault-debugging rate, as

$$m_n = a(1 - (1 - \delta b_f)^n) \tag{17}$$

An alternate formulation of the model, where the fault-debugging rate per undebugged fault, b_{n+1}, is a non-decreasing S-shape curve that captures the learning-process phenomenon of the debugging-team, as

$$b_{n+1} = \frac{b}{1 + \beta(1 - \delta b)^{n+1}} \tag{18}$$

Substituting (18) in (15) and solving using the PGF computational technique in terms of the initial condition that at $n = 0$, $m_n = 0$, we can obtain the closed-form exact solution as

$$m_n = a\frac{1 - (1 - \delta b)^n}{1 + \beta(1 - \delta b)^n} \tag{19}$$

Setting $b = b_f$ and $\beta = \frac{b_f - b_i}{b_i}$, it noticed that (19) and (16) are identical.

The fault-debugging rate for the above-specified model is a logistic function and $b_{n+1} \to b$ as $n \to \infty$. here β is the learning factor, and it represents the skill and experience gained by the debuggers during debugging. If $\beta = 0$, then $b_{n+1} = b$, that is, constant. The fault-debugging intensity function can be obtained as

$$\lambda_n = \frac{m_{n+1} - m_n}{\delta} = \frac{ab}{1 + \beta(1 - \delta b)^n}\,\frac{(1 + \beta)(1 - \delta b)^n}{1 + \beta(1 - \delta b)^n} \tag{20}$$

The equivalent continuous time-space model [1], [16], [31] corresponding to (19) or (16) is

$$m_t = a\frac{1 - (1 - e^{-bt})}{1 + \beta(1 - e^{-bt})} \tag{21}$$

which derived as a limiting case of discrete time-space model substituting $t = n\delta$, $\lim_{x\to 0}(1 + x)^{1/x} = e$, and taking limit $\delta \to 0$.

4) Imperfect Fault-Debugging Inflection S-shaped Model [29], [30]: The primary assumption of the models mentioned above is that the debugged fault is perfectly-debugged with certainty. However, due to human-imperfection and software-complexity, the debugging-team incapable to debug the fault-perfectly, and the debugged fault persist, resulting in a phenomenon known as imperfect fault-debugging. As a result, we write

$$\frac{m_{n+1} - m_n}{\delta} = b_{n+1}(a - m_n)$$

where

$$b_{n+1} = \frac{bp}{1 + \beta(1 - \delta bp)^{n+1}} \tag{22}$$

here p is the probability of perfectly debugging the fault.

Solving (22), using the PGF computational technique in terms of the initial condition that at $n = 0$, $m_n = 0$, we can obtain the closed-form exact solution as

$$m_n = a\frac{1 - (1 - \delta bp)^n}{1 + \beta(1 - \delta bp)^n} \tag{23}$$

The equivalent continuous time-space model [1], [32] corresponding to (23) is

$$m_t = a\frac{1 - e^{-bpt}}{1 + \beta(1 - e^{-bpt})} \tag{24}$$

which derived as a limiting case of discrete time-space model substituting $t = n\delta$, $\lim_{x\to 0}(1 + x)^{1/x} = e$, and taking limit $\delta \to 0$.

In the following section, the approach is extended to address the concept of the change-point.

B. Model Formulation

In addition to the software-reliability models above, several models assume a constant or monotonically increasing fault-debugging rate. However, in reality, as the debugging progress, the fault-debugging rate function may be changed at some point in time. The change point's position need not be estimated; it can be judged from the plotted graph of the actual software-reliability data or using a change point analyzer tool. To incorporate the change-point concept in (26), both fault-debugging rate and probability of perfect fault-debugging before and after the change-point τ, are subject to change. Accordingly, we write

$$b_{n+1} = \begin{cases} \frac{b_1 p_1}{1+\beta(1-\delta b_1 p_1)^{n+1}} & \text{when } 0 \le n < \tau \\ \frac{b_2 p_2}{1+\beta(1-\delta b_2 p_2)^{n+1}} & \text{when } n \ge \tau \end{cases} \tag{25}$$

here $b_1(b_2)$ represent the fault-debugging rates before(after) the change-point, $p_1(p_2)$ represent the probability of perfect fault-debugging before(after) the change-point, and the change-point τ represents the computer test-run number from whose execution onwards change in the fault-debugging rate is noticed.

Substituting (25) in (22) and solving using the PGF computational technique in terms of the initial conditions at $n = 0$, $m_n = 0$ and at $n = \tau$, $m_n = m_\tau$ respectively, we can obtain the closed-form exact solution as

$$m_n = \begin{cases} a\frac{1-\alpha_1^n}{1+\beta\alpha_1^n} & \text{when } 0 \leq n < \tau \\ a\frac{1-(1+\beta)(1+\beta\alpha_2^\tau)(\alpha_1^\tau\alpha_2^{n-\tau})}{(1+\beta\alpha_1^\tau)(1+\beta\alpha_2^n)} & \text{when } n \geq \tau \end{cases} \quad (26)$$

here $\alpha_1 = 1 - \delta b_1 p_1$ and $\alpha_2 = 1 - \delta b_2 p_2$

The proposed modelling approach given by (26) integrates the joint effects of change-point concept and learning of the debugging personnel in an imperfect fault-debugging environment into software-reliability modelling.

III. Data Analyses and Parameter Estimation Techniques

To evaluate the progress of the debugging-process during the testing phase of the SDLC, reliability trend analysis is employed. Therefore, before applying the model, it is reasonable to decide whether the dataset exhibits software-reliability growth. In other words, assessing reliability is pointless if the software-reliability dataset during debugging does not exhibit growth. The trend tests that are widely adopted with f fault debugged are [2], [11], [33]:

- Arithmetic Average Test.

$$\rho(f) = \frac{1}{f}\sum_{i=1}^{f} n_i \quad (27)$$

Decreasing values promote reliability growth.

- Laplace Test.

$$\upsilon(f) = \frac{\sum_{i=1}^{f}(i-1)n_i - \frac{f-1}{2}\sum_{i=1}^{f} n_i}{\sqrt{\frac{f^2-1}{12}\sum_{i=1}^{f} n_i}} \quad (28)$$

Negative values promote reliability growth.

To estimate the unknown parameters of the software-reliability models, either the maximum likelihood estimate (abbreviated as MLE) or the least-square estimate (abbreviated as LSE) methods is employed. Software reliability dataset can be collected during debugging in the form of ordered pairs (n_i, y_i),$(i = 0, 1, 2, \ldots, f)$ where y_i is the cumulative number of faults debugged by n_i computer test-run $(0 < n_1 < n_2 < \ldots < n_f)$. Table I tabulated the software-reliability dataset used. The dataset had been obtained in twenty weeks of debugging release 1 of Tandem computer project, and 100 faults were debugged [34]. In other words, the dataset consisted of 20 data pairs (n_i, y_i), $(i = 0, 1, 2, \ldots, 20; n_{20} = 20; y_{20} = 100)$. To fit and estimate the parameters of the models, all data points have been utilized.

TABLE I. Software-Reliability Dataset [34]

n_i	y_i	n_i	y_i	n_i	y_i	n_i	y_i
1	16	6	49	11	81	16	98
2	24	7	54	12	86	17	99
3	27	8	58	13	90	18	100
4	33	9	69	14	93	19	100
5	41	10	75	15	96	20	100

The graphical plot of the dataset reveals a change-point occurs in the 11th week of debugging, that is, $\tau = 11$.

The Likelihood function L for the unknown parameters with the MVF, that is, m_n takes on the form:

$$L = \prod_{i=1}^{f} \frac{(m_{n_{(i)}} - m_{n_{(i-1)}})^{x_i - x_{i-1}}}{(x_i - x_{i-1})!} e^{-(m_{n_{(i)}} - m_{n_{(i-1)}})} \quad (29)$$

The statistical package for social sciences (abbreviated as SPSS) is applied to estimate the software-reliability model's parameters for quicker and more precise computations.

IV. Model Validation and Comparison Criteria

The credibility of the stochastic software-reliability model is decided by its descriptive and predictive power capabilities [1]–[3], [5], [8], [16], [35].

A. *Goodness-of-Fit Criteria*

To test the validity of the proposed modelling approach in (26), we evaluate three goodness-of-fit test indices based on a real software-reliability dataset [34]. The goodness-of-fit test indices adopted for the purpose are [1]–[3]:

- Sum of squared error (abbreviated as SSE). It assesses the dispersion between estimated values $\widehat{m_{n_i}}$ and the actual data y_i of the dependent variable.

$$SSE = \sum_{i=1}^{f} (\widehat{m_{n_i}} - y_i)^2 \quad (30)$$

- The Akaike information criterion (abbreviated as AIC).

$$AIC = -2logL + 2N \quad (31)$$

here N is the number of parameters of a model and the likelihood function L for the parameters with the MVF in (29).

- Root mean square prediction error (abbreviated as RMSPE).

$$RMSPE = \sqrt{\sum_{i=1}^{f} (Bias^2 + Variation^2)} \quad (32)$$

where

$$Bias = \frac{1}{f}\sum_{i=1}^{f} (\widehat{m_{n_i}} - y_i)$$

$$Variation = \sqrt{\frac{1}{f-1}\sum_{i=1}^{f} ((\widehat{m_{n_i}} - y_i) - Bias)^2}$$

- Coefficient of multiple determination (abbreviated as R^2).

$$R^2 = 1 - \frac{\sum_{i=1}^{f} (\widehat{m_{n_i}} - y_i)^2}{\sum_{i=1}^{f} (y_i - \sum_{i=1}^{f} y_i)^2} \quad (33)$$

It should be noted that the smaller value of SSE, AIC and RMSE, indicates the better the fit of the model. On the contrary, the larger the value of R^2 indicates better the fit of the model.

B. Predictive Validity Criterion

The predictive validity of the software-reliability model is defined as its ability to predict the future behavior of the debugging process based on its current and past behavior [35]. Suppose we have corrected x_f faults after the execution of the last computer test-run n_f. First, we utilize the software-reliability dataset up to a computer test-run say $n_e(\leq n_f)$ to obtain the MVF, that is, m_{n_f}. Then, put estimated values in m_n to obtain the number of faults corrected by n_f. Next, we compare it with the actually corrected number x_f. Last, his method is iterated for different values of n_e. The predicted relative error (abbreviated as PRE) is given as

$$PRE = \frac{(\widehat{m_{n_f}} - x_f)}{x_f} \tag{34}$$

When the PRE value is positive/negative, the model overestimates or underestimates the future debugging phenomenon. Acceptable values are within the range $\pm 10\%$ [1], [2], [16].

C. Software-Reliability Evaluation Measures

Based on Section II, the following quantitative assessment are derived [1]–[3], [26]:

- Number of residual faults. Let $\xi(t)$ denotes the number of residual faults after the executing of the computer test-run n^{th}, then
$$\xi_n = a_\infty - a_n \tag{35}$$
here a_∞ denotes the number of faults eventually corrected.
- Software-reliability. The probability of no faults debugged between the n^{th} and $(n+n_\circ)^{th}$ computer test-run, given that x_d faults have been debugged by the n^{th} computer test-run, is
$$R(n_\circ \mid n) = e^{-(m(n+n_\circ)-m(n))} \tag{36}$$
here $n_\circ(n_\circ \geq 0)$ is the mission time.

V. Dataset Analyses and Model Comparison

To validate the proposed modelling approach, we perform the following dataset analyses and model comparisons. First, we use both of the trend tests presented in Section III to determine whether or not the given software-reliability dataset promotes reliability growth. The parameters of the models under comparison [1], [3], [16], [26], [30] including the proposed modelling approach given in (26) are then estimated using SPSS. Then we perform the goodness-of-fit test described in Section IV-A. Based on the obtained results, we conduct a comparative assessments of the models under comparison. Finally, for the proposed modelling approach, we perform the predictive-validity test described in Section IV-B and estimate the software-reliability evaluation measures defined in Section IV-C.

It is worth noting that, while formulating the proposed modelling approach, we set $\delta = 1$ for simplification and $\tau = 11$ by following the stochastic-behavior of debugging phenomenon on the basis of the adopted dataset.

The models under comparison are given below:

- Exponential model [3], [26] given in (6).
- Delayed S-shaped model [1], [16] given in (10).
- Inflection S-shaped model [1], [30] given in (19).
- Imperfect fault-debugging inflection S-shaped model [1], [30] given in (23).
- The proposed modelling approach given in (26).

A. Software-Reliability Trend Analysis

The fitting result of the reliability dataset for arithmetic average and Laplace trend tests are demonstrated in Fig. 1 and 2, respectively. It is quite clear that the arithmetic average and the Laplace trend factor values are decreasing and entirely negative from the beginning, respectively. Negative Laplace factor values indicate that more faults were debugged in the first half of the debugging time, indicating an increase in reliability. Thus, they point to growth in reliability. Consequently, the dataset is suitable for applying software-reliability models.

B. Goodness-of-Fit Analysis

Table II shows the estimated parameters of the models under comparison using the statistical package statistical package for social sciences (abbreviated as SPSS) and their goodness-of-fit test indices. It can be seen that, when compared to all other models using the SSE, AIC, RMSPE, and R^2 indices, the proposed modelling approach has the lowest SSE, AIC and RMSPE values and the highest R^2 value. Therefore, among the models under comparison, the proposed modelling approach outperformed the others. The enhancement in the performance of the proposed modelling approach is attributed to the inclusion of the change-point concept in modelling software-reliability. These results agree with the findings of previously published works [1], [25].

It is worth noting that the proposed modelling approach's estimated value of a is the closest to the number of debugged faults actually present in the software, and the fault-debugging rates $b_2 < b_1$ clearly show a varying trend that first increases and then decreases following the change-point with an S-shaped varying trend. The latter finding is attributed to the presence of imperfect fault-debugging and learning-process phenomena.

Fig. 3 and 4 show the fitting results of the noncumulative and cumulative reliability dataset for the proposed modelling approach. It is quite clear that the proposed modelling approach fits the dataset perfectly.

C. Predictive Validity Analysis

To estimate the proposed modelling approach parameters, the software-reliability dataset is truncated into several parts. It is noticed that the PRE values differ from one truncation to the next. As a result, Fig. 5 depicts the fitting result of the predictive-validity of the proposed modelling approach.

It is noticed that the proposed modelling approach overestimates the debugging-process from the truncated n_e (70% approx.) onwards. Therefore, the dataset is truncated at t_e

TABLE II. Parameter-Estimation and Comparison-Criteria Results

Models Under Comparison	Parameter-Estimation						Comparison-Criteria			
	a	$b_1(b)$	$p_1(p)$	b_2	p_2	β	SSE	AIC	$RMSPE$	R^2
Model due to [3], [26]	130	.080	—	—	—	—	233	1789	3.449	.986
Model due to [1], [16]	106	.217	—	—	—	—	357	1796	4.329	.978
Model due to [1], [30]	111	.158	—	—	—	1.21	180	1787	3.073	.989
Model due to [1], [30]	111	.158	1	—	—	1.21	180	1787	3.073	.989
Proposed given in (26)	105	.474	1	.357	.675	5.92	35	1780	1.352	.998

— The parameter is not part of the corresponding model.

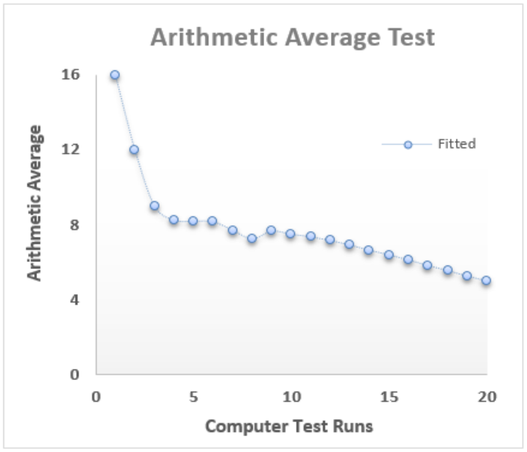

Fig. 1. The Fitting Result of the Reliability Dataset for Arithmetic Average Test Data.

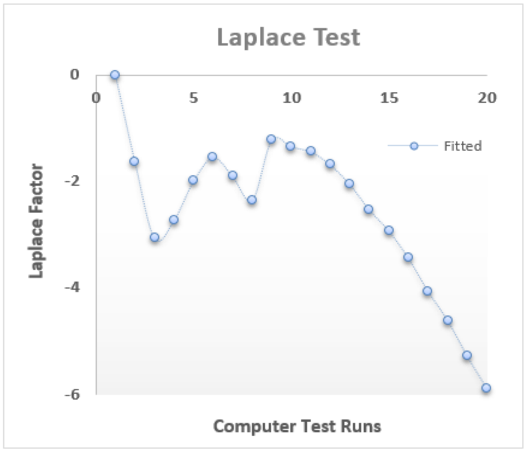

Fig. 2. The Fitting Result of the Reliability Dataset for Laplace Test Data.

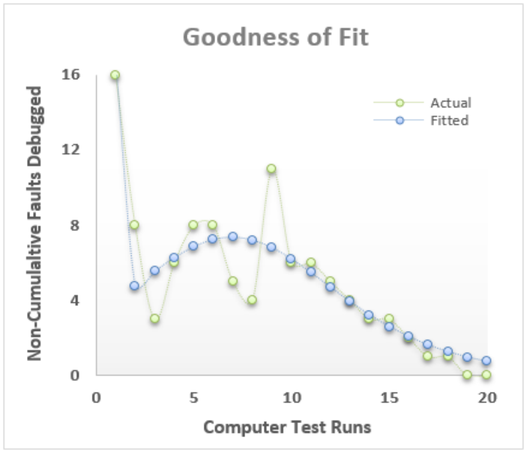

Fig. 3. The Fitting Result of the Noncumulative Reliability Dataset for the Proposed Modelling Approach.

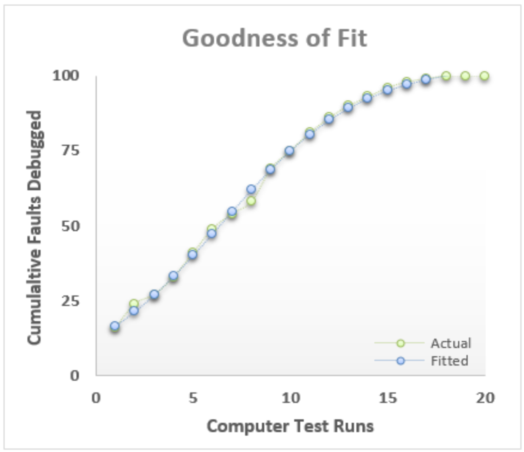

Fig. 4. The The Fitting Result of the Cumulative Reliability Dataset for the Proposed Modelling Approach.

(60% approx.) to obtain the whole dataset. The fitting result of the retrodictive and predictive capabilities is shown in Fig. 6. The points below n_e (marked by the intersection of the horizontal line with the curve) show the retrodictive-capability while the points above n_e show the predictive-capability of the proposed modelling approach.

The proposed modelling approach reveals that about 60% of the debugging-time is enough to predict the debugging-future process's behavior satisfactorily.

D. Software-Reliability Quantitative Measures Analysis

Fig. 7 depicts the fitting result of the number of residual faults for the proposed modelling approach. It is quite clear that the proposed modelling approach fits the dataset perfectly. Following the change-point, every three faults were debugged, one of which was imperfectly debugged according to the parameters' estimated values in Table II. Fig. 8 depicts the fitting result of the estimated software-reliability (three cases: when the mission time is one, two, and three days respectively) for the proposed modelling approach. It is noticeable that the proposed modelling approach reveals that software-reliability

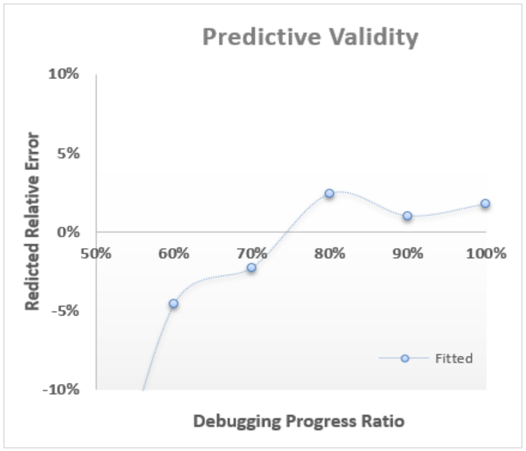

Fig. 5. The Fitting Result of the Predictive Validity for the Proposed Modelling Approach.

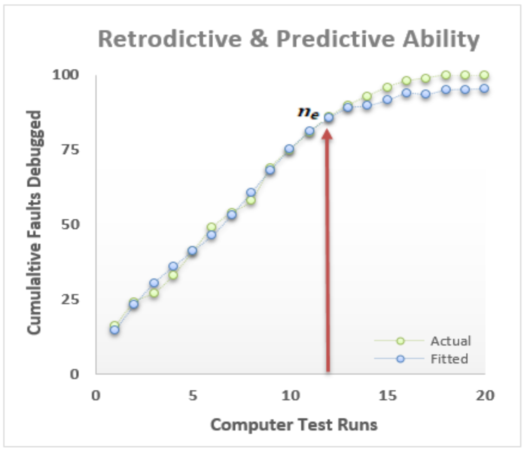

Fig. 6. The Fitting Result of the Retrodictive and Predictive Capabilities for the Proposed Modelling Approach.

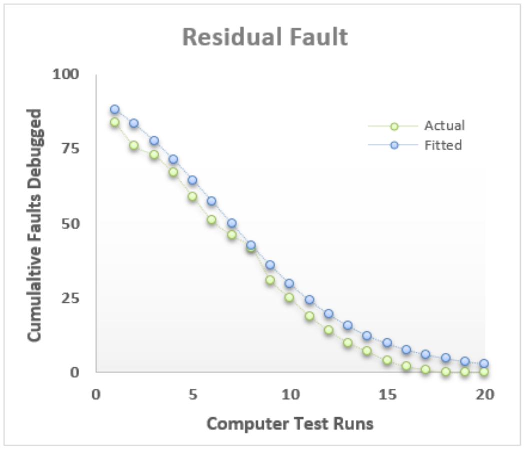

Fig. 7. The Fitting Result of the Number of Residual Faults for the Proposed Modelling Approach.

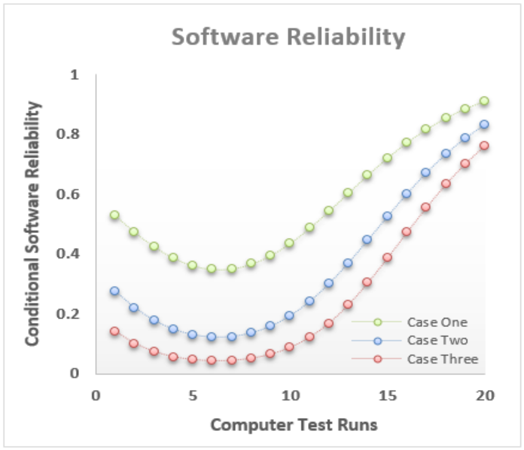

Fig. 8. The Fitting Result of the Estimated Software-Reliability (three cases) for the Proposed Modelling Approach.

monotonically increases after the change-point in all cases.

These types of information provided by the proposed modelling approach enable the developer to decide how best to allocate resources during the software testing and maintenance phases of the SDLC.

VI. Conclusions

Quantitative measures provided by software-reliability models play a pivotal role in the decision-making process during testing and fault debugging-process. To some extent, the concept of change-point is novel in the discrete time-space software-reliability modelling approach. The NHPP type of discrete time-space software-reliability modelling is well-documented with well-established concepts in the software-reliability engineering literature and can be used to describe the stochastic-behavior of the fault-debugging process.

The proposed discrete time-space NHPP based stochastic software-reliability model integrates the debugger's learning phenomenon with a single change-point under an imperfect fault-debugging environment. The results explained through a numerical example in the tables and figures provided in Section V are encouraging compared with other well-established models built under imperfect and perfect fault-debugging environments. The conducted numerical example concludes that incorporating the concept of change-point into software-reliability modelling yields the most promising results concerning the descriptive and predictive performances of the proposed modelling approach and software-reliability quantitative measures. As a result, the paper demonstrates a high level of numerical agreement between the results of the proposed modelling approach obtained here and other results presented in the literature.

Finally, the paper shows how starting with fundamental assumptions, the discrete time-space NHPP based modelling approach is becoming more useful with the inclusion of the learning process in debugging and introducing the change-point concept in an imperfect fault-debugging environment. However, we confined ourselves to introducing just a single change-point, and we did not consider the debugging effort expenditures. Hence, it is essential to track and account for the reliability growth concerning the costs spent on software debugging. Studying the effect of multiple change-point occurrences and software debugging efforts on reliability growth is an ongoing and future research effort.

Acknowledgment

We thank the two anonymous reviewers whose valuable, constructive and insightful comments helped improve and clarify the paper.

References

[1] P. K. Kapur, H. Pham, A. Gupta and P. C. Jha, Software Reliability Assessment with OR Applications, Springer, 2011.

[2] O. Shatnawi, ”An integrated framework for developing discrete-time modelling in software reliability engineering,” Quality and Reliability Engineering International, vol. 32, pp. 2925–2943, 2016.

[3] S. Yamada, Software Reliability Modeling: Fundamentals and Applications, Springer, 2014.

[4] P. K. Kapur, S. K. Khatri, A. Tickoo and O. Shatnawi, ”Release time determination depending on number of test runs using multi attribute utility theory,” International Journal of System Assurance Engineering and Management, vol. 5, pp. 186–194, 2014.

[5] M. Xie, Software Reliability Modeling, World Scientific, 1991.

[6] D. H. Lee, I. H. Chang and H. Pham, ”Software reliability model with dependent failures and SPRT,” Mathematics, vol. 8, 1366 pages, 2020.

[7] N. Qallab and O. Shatnawi, ”Modelling software reliability growth phenomenon in distributed development environment,” International Journal of Scientific & Technology Research, vol. 8, pp. 2176–2184, 2019.

[8] H. Pham, System Software Reliability, Springer, 2007.

[9] P. K. Kapur, A. G. Aggarwal, O. Shatnawi and R. Kumar, ”On the development of unified scheme for discrete software reliability growth modeling,” International Journal of Reliability, Quality and Safety Engineering, vol. 17, pp. 245–260, 2010.

[10] T. Pham and H. Pham, ”A generalized software reliability model with stochastic fault-detection rate,” Annals of Operations Research, vol. 227, pp. 83–93, 2019.

[11] O. Shatnawi, ”Measuring commercial software operational reliability: an interdisciplinary modelling approach,” Eksploatacja i Niezawodność – Maintenance & Reliability, vol. 16, pp. 585–594, 2014.

[12] P. K. Kapur, O. Singh, S. K. Khatri and A.K. Verma, Strategic System Assurance and Business Analytics, Springer, 2020.

[13] K. Deep and M. Jain, S. Salhi, Performance Prediction and Analytics of Fuzzy, Reliability and Queuing Models: Theory and Applications, Springer, 2018.

[14] D. D. Hanagal and N. N. Bhalerao, Software Reliability Growth Models, Springer, 2021.

[15] O. Shatnawi, ”Testing-effort dependent software reliability model for distributed systems, International Journal of Distributed Systems and Technologies, vol. 4, pp. 1–14, 2013.

[16] P. K. Kapur, S. Kumar and R. B. Garg, Contributions to Hardware and Software Reliability, World Scientific, 1999.

[17] M. Zhao, ”Change-point problems In software and hardware reliability,” Communications in Statistics Theory and Methods, vol. 22, pp. 757–768, 1993.

[18] V. Nagaraju, L. Fiondella and T. Wandji, ”A heterogeneous single changepoint software reliability growth model framework,” Software Testing, Verification and Reliability, vol. 29, e1717, 2019.

[19] Y. Minamino, S. Inoue and S. Yamada, ”NHPP-based change-point modeling for software reliability assessment and its application to software development management,” Annals of Operations Research, vol. 244, pp. 85–101, 2016.

[20] P. K. Kapur, A. Gupta, O. Shatnawi and V.S.S. Yadavalli, ”Testing effort control using flexible software reliability growth Model with change point,” International Journal of Performability Engineering, vol. 2, pp. 245–262, 2006.

[21] S. Inoue, J. Ikeda and S. Yamda, ”Bivariate change-point modeling for software reliability assessment with uncertainty of testing-environment factor,” Annals of Operations Research, vol. 244, pp. 209–220, 2016.

[22] S. Chatterjee and A. Shukla, ”Change point–based software reliability model under imperfect debugging with revised concept of fault dependency,” Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, vol. 230, pp. 579–597, 2016.

[23] V. Nagaraju, L. Fiondella and T. Wandji, ”A heterogeneous single changepoint software reliability growth model framework,” Software Testing, Verification and Reliability, vol. 29, e1717, 2019.

[24] O. Shatnawi, P. K. Kapur and M. T. Shatnawi, ”Release policy, change-point concept, and effort control through discrete-time imperfect software reliability modelling,” International Journal of Computer Applications, vol. 137, pp. 17–25, 2016.

[25] D. N. Goswami, S. K. Khatri, R. Kapur, ”Discrete software reliability growth modeling for errors of different severity incorporating change-point concept,” International Journal of Automation and computing, vol. 4, pp. 396–405, 2007.

[26] S. Yamada and S. Osaki, ”Discrete software reliability growth models,” Applied stochastic models and data analysis, vol. 1, pp. 65–77, 1985.

[27] A. L. Goel and K. Okumoto, ”Time-dependent error-detection rate model for software reliability and other performance measures,” IEEE transactions on Reliability, vol. 34, pp. 206–211, 1979.

[28] S. Yamada, M. Ohba and S. Osaki, ”S-shaped reliability growth modeling for software error detection”, IEEE Transactions on reliability, vol. 32, 475–484, 1983.

[29] O. Shatnawi and M. T. Shatnawi, ”Mathematical modelling in software reliability,” Proceedings of the International Conference on Computational & Information Science, University of Houston, USA, 2009, pp. 425–430.

[30] O. Shatnawi, ”Discrete time NHPP models for software reliability growth phenomenon,” International Arab Journal of Information Technology, vol. 6, pp. 124–131, 2009.

[31] M. Ohba, ”Software reliability analysis models,” IBM Journal of research and Development, vol. 28, pp. 428–443, 1984.

[32] K. Edris and O. Shatnawi, ”The Pham Nordmann Zhang (PNZ) software reliability model revisited,” Proceedings of the 10$^{\text{th}}$ IASTED International Conference on Software Engineering, Innsbruck, Austria, 2011, pp.15–17.

[33] K. Kanoun, M. Kaniche and J. P. Laprie, ”Qualitative and quantitative reliability assessment,” IEEE Software, vol. 14, pp. 77–87, 1997.

[34] A. Wood, ”Predicting software reliability,” IEEE Computers, vol. 29, pp. 69–77, 1996.

[35] J. D. Musa, Software Reliability Engineering: More Reliable Software, Faster and Cheaper, McGraw-Hill, 2004.