Article citation info:
SHATNAWI O. Measuring commercial software operational reliability: an interdisciplinary modelling approach. Eksploatacja i Niezawodnosc – Maintenance and Reliability 2014; 16 (4): 585–594.

Omar SHATNAWI

MEASURING COMMERCIAL SOFTWARE OPERATIONAL RELIABILITY: AN INTERDISCIPLINARY MODELLING APPROACH

POMIAR NIEZAWODNOŚCI EKSPLOATACYJNEJ OPROGRAMOWANIA KOMERCYJNEGO: INTERDYSCYPLINARNE PODEJŚCIE DO MODELOWANIA

In the software reliability engineering (SRE) literature, few attempts have been made to model the failure phenomenon of commercial software during its operational use. One of the reasons can be attributed to the inability of software engineers to measure the growth in usage of commercial software while it is in the market. It is unlike the testing phase where resources follow a definite pattern. In this paper, an attempt has been made to model the software reliability growth linking it to the number of users. Since the number of instructions executed depends on the number of users. The number of users is estimated through an innovation diffusion model of marketing. Once the estimated value is known, the rate at which instructions are executed can be found. The intensity with which failures would be reported depends upon this value. To model the failure observation or defect removal phenomena, a non-homogenous Poisson process (NHPP) based software reliability models developed in the literature have been employed. Software reliability models are most often used for reliability projection when development work is complete and before the software is shipped to customers. They can also be used to model the failure pattern or the defect arrival pattern in the field and thereby provide valuable input to maintenance planning. Numerical example with real software field reliability data is presented to illustrate descriptive and predictive performance as well as to show practical applications of the proposed models.

Keywords: *software reliability engineering, software reliability models, non-homogenous Poisson process, imperfect debugging, commercial software usage, innovation diffusion.*

Literatura dotycząca inżynierii niezawodności oprogramowania, podejmuje zaledwie nieliczne próby modelowania zjawiska uszkodzenia oprogramowania komercyjnego w trakcie jego eksploatacji. Jednym z powodów może być to, iż programiści nie są w stanie zmierzyć wzrostu użytkowania oprogramowania komercyjnego w trakcie obrotu handlowego tego typu oprogramowaniem. Etap ten różni się bowiem od fazy testowania, gdzie zasoby funkcjonują według określonego wzorca. W niniejszej pracy podjęto próbę stworzenia modelu wzrostu niezawodności oprogramowania łącząc to pojęcie z pojęciem liczby użytkowników, jako że liczba wykonywanych poleceń zależy właśnie od liczby użytkowników. Liczbę użytkowników szacuje się na podstawie modelu marketingu opartego na dyfuzji innowacji. Gdy szacowana wartość jest już znana, można określić częstość wykonywania poleceń. Intensywność zgłaszania uszkodzeń zależy od tej wartości. Do modelowania zjawisk zaobserwowania uszkodzenia lub usunięcia usterki zastosowano opracowane wcześniej w literaturze modele niezawodności oprogramowania oparte na niejednorodnym procesie Poissona (NHPP). Modele niezawodności oprogramowania są najczęściej wykorzystywane do projektowania niezawodności już po zakończeniu prac rozwojowych, ale zanim jeszcze oprogramowanie dotrze do klientów. Mogą być również stosowane do modelowania wzorców uszkodzeń lub wzorców występowania usterek w trakcie eksploatacji, stanowiąc tym samym cenny wkład do planowania czynności konserwacyjnych. Przykład liczbowy uwzględniający dane z eksploatacji rzeczywistego oprogramowania ilustruje opisowe i predykcyjne możliwości proponowanych modeli, jak również pokazuje, jak można je stosować w praktyce.

Słowa kluczowe: *inżynieria niezawodności oprogramowania, modele niezawodności oprogramowania, niejednorodny proces Poissona, niedoskonałe debugowanie, użytkowanie oprogramowania komercyjnego, dyfuzja innowacji.*

1. Introduction

Software reliability is defined as the probability of failure–free software operation for a specified period of time (American National Standards Institute – ANSI). It quantifies the failures of software systems and is the key factor in software quality [19]. It is also a major subject of Software Reliability Engineering (SRE) – a discipline which quantitatively studies the operational behavior of software systems with respect to the reliability requirements of the user. The quantitative study of software systems concerning reliability involves software reliability measurements. Measurement of software reliability includes two activities, i.e., software reliability estimation and software reliability prediction. Software reliability modelsare used to measure a software product's reliability or to estimate the number of latent defects when it is available to the customers. Such an estimate is important for two reasons: 1) as an objective statement of the quality of the product and 2) for resource planning for the software maintenance phase [9].

Research has been conducted in software reliability engineering over the past three decades and many software reliability models have been proposed [4, 12, 19, 20, 23, 29, 30]. The pioneering attempt in non-homogenous Poisson process (NHPP) based on software reliability model was the exponential model [7]. The model describes the failure/removal phenomenon by an exponential curve. There are also software reliability models that describe either S-shaped curves or a mixture of exponential and S-shaped curves (i.e., flexible). Some of the important contributions of these type of models are due to[11, 21, 32] etc. In most of these models it is assumed that whenever an attempt is made to remove a defect, it is removed with certainty i.e., a case of perfect debugging. But the debugging activity is not always

perfect because of number of factors like tester's skill/expertise etc. In practical software development scenario, the number of failures observed/detected may not be necessarily same as the number of defect removed/corrected. Kapur and Garg [11] have discussed in their defect removal phenomenon model that as testing grows and testing and debugging team gains experience, additional numbers of defects are removed without them causing any failure. The testing and debugging, however, may not be able to remove/correct defect perfectly on observation/detection of a failure and the original defect may remain leading to a phenomenon known as imperfect debugging, or replaced by another defect resulting in defect generation. In case of imperfect debugging the defect-content of the software is not changed, but because of incomplete understanding of the software, the original detected defect is not removed perfectly. But in case of defect generation, the total defect-content increases as the testing and debugging progresses because new defects are introduced in the system while removing the old original defects [5, 14, 15, 26].Models due to [22, 33] are defect generation models applied on the exponential model [7] and have been also named as imperfect debugging models. Kapur et al. [12] introduced the imperfect defect debugging in exponential model [7]. They assumed that the defect detection rate per remaining defects is reduced due to imperfect defect debugging. Thus the number of failures observed/detected by time infinity is more than the initial defect-content. Although these two models describe the imperfect debugging phenomenon yet the software reliability growth curve of these models is always exponential. Moreover, they assume that the probability of imperfect debugging is independent of the testing time. Thus, they ignore the role of the learning process during the testing phase by not accounting for the experience gained with the progress of software testing. Zhang et al. [34] proposed a testing efficiency model which includes both imperfect defect debugging and defect generation, modeling it on the number of failures experienced, however both imperfect debugging and defect generation are actually seen during defect removal. Recently, Kapur et al. [15] proposed a flexible software reliability model with imperfect defect debugging and defect generation using a logistic function for defect detection rate, which reflects the efficiency of the testing and debugging team.

Very few attempts have been made to model the failure phenomenon of software product during its operational use. One of the reasons for this can be attributed to the inability of software engineers to measure the growth in usage of software while it is in the market. It is unlike the testing phase where testing-effort follows a definite pattern. Kenney [17] developed a calendar-time model for a multi-release product using Trachtenberg's [28] general theory of software reliability. He has assumed a power function to represent the usage rate of the software. Though he argues that the rate at which the software product is used is dependent upon the number of its users, the model proposed by him fails to capture the growth in number of users of the software. To capture the growth in number of end-users of the software, Kapur et al. [13] incorporatesa model from marketing to account for usage in the operational phase as for the commercially used software, number of instructions executed depends on the number of users.

The rest of this paper is structured as follows. Section 2 provides an interdisciplinary modeling approach that combines the subject software reliability engineering and marketing. Sections 3 and 4presentparameter estimation techniquesand filed software reliability data. In Sections 5 and 6, data analyses techniques,and model validation and comparison criteria are discussed. Filed software reliability data analyses and model comparisons discussed in Section 7 and Section 8 concludes the paper with some general remarks.

2. Interdisciplinary modelling approach

Study of a system in isolation can be easier but may not provide the optimal results. On the other hand, the theory of a single discipline may prove to be inadequate in explaining the dynamic interactions with other systems. Hence there is a need for interdisciplinary approach. The distinguishing feature of modern science has been the increasing interweaving of formally separated disciplines. Mathematical modeling is a collection of tools that cannot be put under one single discipline. They have been facilitating interdisciplinary studies of many complex situations. Mathematical modeling in marketing started in 1950s. They have been applied to measure the effectiveness of promotional campaign, brand switching behaviour of consumers, success of a new product, market risk analysis, etc. Mathematical models have been proposed for testing-effort [1, 18, 25, 31] but they are not suitable for measuring usage of software in the market. The intensity with which failures would manifest during the operational use is dependent upon the number of times the software is used and not much has been done in the literature for the situation [3]. Many interdisciplinary studies as production management and financial management have also been carried out. But few attempts have been made at including reliability models, though quality is a very important attribute of a successful product.In the software reliability engineering literature, few attempts have been made to include marketing parameters for evaluating the operational reliability. One attempt has used a modified version of innovation diffusion model [2] to estimate the number of licensed users as well as users of pirated copies of the software [6].

For a reliable estimate of the growth with time in number of users who use a particular software release product during operational phase, we have employed a mathematical model developed in the discipline of marketing management [2]. The employed model can be used to account for usage in the operational phase as for the commercially used software, number of instructions executed depends on the number of users. The usage function so defined can also take care of increasing, linearly decreasing trends as a function of time, which implies slow start but gain in growth rate. A big beginning and tailing off in the usage growth.The usage function is estimated through innovation diffusion model of marketing. Such an interdisciplinary modeling approach that combines the subjectssoftware engineering and marketing has been attempted for the first time [16].

1. Software is subject to failures during execution caused by defects remaining in the software
2. Software failure occurrence or defect removal phenomena follows an NHPP with .
3. Software usage is used as a basis for failure rate.
4. The number of failures experienced during operation is dependent upon the number of instructions executed.
5. The number of instructions executed is a function of the number of users.
6. The number of users is a function of time.

The following notations are used for the mathematical formulation purpose:

- m Expected number of failures experienced in the time interval $(0, t]$
- e Expected number of instructions executed on the software in $(0, t]$
- W Expected number of software users in $(0, t]$ and $\partial W/\partial t = w_t$
- a Expected number of defects lying dormant in software
- b Proportionality constant denotes the rate at which remaining defects cause failures
- p Probability of defect removal on a failure
- α Rate at which the defect may be introduced during the debugging process
- σ Rate at which instructions are executed

β Constant representing learning parameter in logistic function
τ Expected number of potential users in the population
μ Coefficient of external (mass media) influence (i.e., innovation rate)
η Coefficient of internal (inter-personal) influence (i.e., imitation rate)
k,γ Constants.

2.1. Development

We model the number of reported failures time t as a pure birth counting process $(N_t)_{t\geq 0}$, or more specifically, a NHPP. A pure birth counting process $(N_t)_{t\geq 0}$ is a NHPP with intensity function λ_t, for all $t\geq 0$, if it satisfies the following properties:

1) $N_{t=0}=0$
2) $(N_t)_{t\geq 0}$ has independent increments. This implies that for any $t_i>t_j>t_k>t_l$ the random variables $N_{t_j}-N_{t_i}$ and $N_{t_l}-N_{t_k}$ are independent.
3) The random variable $N_{t_j}-N_{t_i}$ has a Poisson distribution with mean $m_{t_j}-m_{t_i}$, for all $0\leq t_i<t_j$. This implies that:

$$\mathbb{P}\left[N_{t_j}-N_{t_i}=k\right]=\frac{\left(m_{t_j}-m_{t_i}\right)^k}{k!}\cdot e^{-\left(m_{t_j}-m_{t_i}\right)}, \text{ for all } k=0,1,\ldots, \quad (1)$$

where $m_t=\int_0^t \lambda_x \cdot d_x$ is the mean value function of the NHPP $[N_t, t\geq 0]$.

The quantity m_t describes the expected number of failures or defect removal up to time t. Because of the underlying assumptions about the failures and number of defects in the software, we assume m_t to be a bounded, strictly increasing function satisfying the boundary conditions $m_{t=0}=0$.

Using the above assumptions the failure occurrence or the defect removal phenomena can be described with respect to time as follows:

$$\lambda_t=\frac{\partial m}{\partial t}=\frac{\partial m}{\partial e}\cdot\frac{\partial e}{\partial W}\cdot\frac{\partial W}{\partial t} \quad (2)$$

Each component on the right hand side of the differential Eq. (2) is individually discussed below.

2.1.1. Modeling the number of failures reported per instructions

The first component of Eq. (2) relates the number of failures experienced during operation with the number of instructions executed. In other words, during testing instructions are executed on the software and the output is matched with the expected results. If there are any discrepancy a failure is said to have occurred. Effort is made to identify and later remove the cause of the failure. The earlier models due to Kapur et al. [13] and Shatnawi [27] have the employed flexible model [11] and the exponential model [7] respectively, for the purpose. However, in the exponential model [7] defects are removed immediately after a software failure is observed, i.e. the time to remove a defect is negligible. But in reality, each observed failure is reported, diagnosed, corrected, and then verified. The time from observation to removal should not be neglected in a practical software testing process. Besides, in the flexible model [11] defects are removed with certainty and no new defect introduced during testing and debugging process. In reality this may not be always true. The corrections may themselves introduce new faults or they may inadvertently create conditions, not previously experienced, that enable other faults to cause failures. This results in situations where the actual fault removals are less than the removal attempts. To address these issues the models due to Yamada et al. [32] and Kapur et al. [15] are selected accordingly.

The first selected software reliability model is the delayed S-shaped model [32] that describes the testing and debugging process as a two-stage process—failure observation and the corresponding defect removal phenomenon. This model can be derived alternatively in one stage as follows:

$$\frac{\partial m_w}{\partial e}=b_w\cdot\left(a-m_w\right), \quad (3)$$

where $b_w=\frac{b^2\cdot W_t}{1+b\cdot W_t}$.

The second selected software reliability model is the testing efficiency model [15] that integrates the effect of imperfect defect debugging and defect generation using a logistic function for the defect detection rate, which reflects the efficiency of the testing and removal team. In this model, the failure intensity satisfies the following differential equation:

$$\frac{\partial m_w}{\partial e}=p\cdot b_w\cdot\left(a_w-m_w\right), \quad (4)$$

where $b_w=\frac{b}{1+\beta\cdot e^{-b\cdot W_t}}$ and $a_w=a+\alpha\cdot m_w$.

2.1.2. Modeling the number of instructions executed per users

The second component of Eq. (2) relates the number of instructions executed with the testing effort or the number of users of the software. For the sake of simplicity we assume it to be constant:

$$\frac{\partial e}{\partial W}=\sigma, \quad (5)$$

2.1.3. Modeling the number of users per unit time

The third component of Eq. (2) relates the growth in number of users with respect to time. Kenny [17] has used the power function to describe the growth in user population who use a particular software release [16]:

$$W_t=\frac{t^{k+1}}{k+1}, \quad (6)$$

W_t here is the number of users of the software in the operational phase at time t.

The function can correctly describe the users growth in terms of a slow start but gain in growth rate, a constant addition of users, or a big beginning and tail off in the usage rate. In the marketing literature, power function is rarely used for the purpose as described above. One of the reasons can be that the parameters of the function are not amenable to interpretations. In models proposed by [13, 27], the growth in number of users (or adopters) with respect to time using is described by the Bass [2] new product diffusion model. In marketing, the diffusion of innovations occurs with every launch of a new type of product, and is widely thought to be influenced by both inter-personal and mass media communication. Bass labelled those who adopt due to external influences innovators, and those who adopt due to internal influences imitators.

Mathematically the relationship is expressed as follows:

$$\frac{\partial W_t}{\partial t}=\mu\cdot\left(\tau-W_t\right)+\eta\cdot\frac{W_t}{a}\cdot\left(\tau-W_t\right), \quad (7)$$

Solving Eq. (7) with the boundary condition $W_{t=0}$=0, we have:

$$W_t = \tau \cdot \frac{1-\mathrm{e}^{-(\mu+\eta)\cdot t}}{1+\frac{\eta}{\mu}\cdot \mathrm{e}^{-(\mu+\eta)\cdot t}}. \quad (8)$$

2.2. Formulation

Substituting Eq. (3) and Eq. (5) in Eq. (2), we have:

$$\frac{\partial m}{\partial t} = p\cdot\frac{b^2\cdot W_t}{1+b\cdot W_t}\cdot(a-m)\cdot\sigma\cdot\frac{\partial W}{\partial t}, \quad (9)$$

Solving Eq. (9) under boundary condition $m_{t=0}$=0, we have the first proposed model as:

$$m_t = a\cdot\left(1-\left(1+b\cdot W_t\right)^{\gamma}\cdot e^{-b\cdot\gamma\cdot W_t}\right), \quad (10)$$

here $p\cdot\sigma=\gamma$.

The failure intensity is given as:

$$\lambda_t = \frac{\partial m}{\partial t} = \frac{a\cdot\gamma\cdot b^2\cdot W_t}{1+b\cdot W_t}\cdot\left(\left(1+b\cdot W_t\right)^{\gamma}\cdot e^{-b\cdot\gamma\cdot W_t}\right), \quad (11)$$

Substituting Eq. (4) and Eq. (5) in Eq. (2), we have:

$$\frac{\partial m}{\partial t} = p\cdot\frac{b}{1+\beta\cdot\mathrm{e}^{-b\cdot W_t}}\cdot\left(a-(1-\alpha)\cdot m\right)\cdot\sigma\cdot\frac{\partial W}{\partial t}, \quad (12)$$

Solving Eq. (12) under boundary condition $m_{t=0}$=0, we have the second proposed model as:

$$m_t = \frac{a}{1-\alpha}\cdot\left[1-\left(\frac{(1+\beta)\cdot e^{-b\cdot W_t}}{1+\beta\cdot e^{-b\cdot W_t}}\right)^{\gamma\cdot(1-\alpha)}\right], \quad (13)$$

here $p\cdot\sigma=\gamma$.

The failure intensity is given as:

$$\lambda_t = \frac{\partial m}{\partial t} = \frac{a\cdot\gamma\cdot b}{1+\beta\cdot e^{-b\cdot W_t}}\cdot\left(\frac{(1+\beta)\cdot e^{-b\cdot W_t}}{1+\beta\cdot e^{-b\cdot W_t}}\right)^{\gamma\cdot(1-\alpha)}, \quad (14)$$

It should be pointed out here that the second proposed model given in Eq. (13) is more general than that of the first proposed model given in Eq. (10) since it includes the effect of defect generation, imperfect defect debugging, and has the models due to [13, 27]as special cases.

3. Parameter estimation techniques

Parameters estimation is of primary concern in software reliability measurement. Software filed reliability data can be collected during operational from the actual operational sites where software is used by its intended users during field tests in the form of failures x_i($-<x_1<x_2<\cdots<x_k$) reported by sites W_i ($W_1<W_2<W_3<\cdots<W_k$) in the time interval $(0,t_i]$ where i=1,2,…,k. Data usage collected during operational use is estimated by the method of least square as follow:

$$\begin{aligned} &\text{minimize } \sum_{i=1}^{k}\left(W_i-\hat{W}\right)^2, \\ &\text{subject to } \quad \hat{W}_k = W_k, \end{aligned} \quad (15)$$

where $\hat{W}_k=W_k$ implies that the estimated value is equal to the actual value.

Using these estimated parameters values, we estimate the parameters in the proposed models given in Eq. (10) and Eq. (13)and the models under comparison given Table 1 by the method of maximum likelihood estimation (MLE). The Likelihood function L for the unknown parameters with the mean value function m_t takes on the form:

$$L\left(parameters|\left(W_i,x_i\right)\right) = \prod_{i=1}^{k}\frac{\left(m_{t_i}-m_{t_i-1}\right)^{x_i-x_{i-1}}}{\left(x_i-x_{i-1}\right)!}e^{-\left(m_{t_i}-m_{t_i-1}\right)}, \quad (16)$$

Taking natural logarithm of (16) we get:

$$\ln L = \sum_{i=1}^{k}\left(x_i-x_{i-1}\right)\ln\left(m_{t_i}-m_{t_i-1}\right)-\left(m_{t_i}-m_{t_i-1}\right)\sum_{i=1}^{k}\ln\left(x_i-x_{i-1}\right) \quad (17)$$

The MLE of the unknown parameters can be obtained by maximizing the likelihood function subject to the parameters constraints.

For faster and accurate calculations, the statistical package for social sciences (SPSS) based on the nonlinear regression technique has been utilized for the estimation of the parameters of the proposed models and the models under comparison. Non-linear regression is a technique of finding a nonlinear model of the relationship between the dependent variable and a set of independent variables. Unlike traditional linear regression, which is restricted to estimating linear models, non-linear regression can estimate models with arbitrary relationships between independent and dependent variables.

Table 1. Models under comparison.

Reference	Software Reliability Model
Model due to [27]	$a\cdot\left[1-e^{-b\cdot\sigma\cdot W_t}\right]$
Model due to [13]	$a\cdot\left[1-\left(\frac{(1+\beta)\cdot e^{-b\cdot W_t}}{1+\beta\cdot e^{-b\cdot W_t}}\right)^{\sigma}\right]$
Proposed given in Eq. (10)	$a\cdot\left[1-\left(1+b\cdot W_t\right)^{\gamma}\cdot e^{-b\cdot\gamma\cdot W_t}\right]$
Proposed given in Eq. (13)	$\frac{a}{1-\alpha}\cdot\left[1-\left(\frac{(1+\beta)\cdot e^{-b\cdot W_t}}{1+\beta\cdot e^{-b\cdot W_t}}\right)^{\gamma\cdot(1-\alpha)}\right]$

4. Filed software reliability data

An actual field data from a larger release of a telecommunications switch software given in Table 2. This data is available in the form (t_i,w_i,x_i)(i=1,2,3,…,140)where w_iis the number of sites reporting failures x_i per time t_i ($t_1<t\_2<t_3<\cdots<t_{140}$), t_{140}=93.5, $W_{140}=\sum_{i=1}^{140}w_i=8109$, and x_{140}=100. Note that the data has been normalized to protect proprietary information. The main effect of normalization on the analysis is one of scaling. Therefore, the analysis of the non-normalized data would be essentially the same [8] and further studied in [27]. Therefore, the data set allows direct comparison with the work of others.

Table 2. Field data from a larger release of a telecommunications switch software. Data is calendar-time, %Cum SW Failures is the percentage of the total number of software failure experienced in the calendar interval reported in the table, %Cum Usage Time is the percentage of the total in service time accumulated over the calendar interval reported, and %Sites is the percentage of sites that have this version of the software release loaded on a given date. Note that the data has been normalized to protect proprietary information [8].

Date	%cum SW Failures	%cum Usage Failures	%Site	Date	%cum SW Failures	%cum Usage Failures	%Site	Date	%cum SW Failures	%cum Usage Failures	%Site	Date	%cum SW Failures	%cum Usage Failures	%Site
1	0	0.0	2	36	35	11.4	73	71	79	49.0	98	106	96	78.8	54
2	1	0.1	2	37	38	12.2	75	72	79	50.2	97	107	96	79.4	53
3	1	0.1	2	38	39	13.1	76	73	80	51.3	95	108	97	80.0	52
4	1	0.1	2	39	40	14.0	78	74	81	52.4	95	109	97	80.6	50
5	1	0.1	2	40	42	14.9	81	75	82	53.4	93	110	97	81.2	48
6	1	0.2	3	41	44	15.8	83	76	83	54.5	91	111	97	81.7	48
7	2	0.2	3	42	45	16.8	85	77	83	55.5	90	112	98	82.3	47
8	2	0.3	4	43	47	17.8	86	78	84	56.6	89	113	98	82.8	46
9	2	0.3	5	44	49	18.8	88	79	85	57.6	88	114	98	83.3	45
10	2	0.4	5	45	50	19.8	90	80	85	58.6	85	115	98	83.9	45
11	3	0.4	6	46	52	20.9	92	81	86	59.6	82	116	98	84.4	45
12	4	0.5	7	47	54	21.9	94	82	86	60.5	80	117	98	84.9	42
13	4	0.6	7	48	55	23.0	95	83	87	61.4	78	118	98	85.3	40
14	5	0.7	9	49	56	24.1	96	84	87	62.3	78	119	98	85.8	39
15	7	0.8	13	50	57	25.2	96	85	88	63.2	76	120	98	86.3	38
16	7	1.0	16	51	58	26.3	95	86	88	64.1	75	121	98	86.6	38
17	8	1.1	16	52	59	27.4	95	87	89	64.9	74	122	98	87.1	37
18	9	1.5	17	53	60	28.5	95	88	89	65.8	72	123	98	87.5	37
19	10	1.7	91	54	60	29.6	96	89	90	66.6	71	124	98	87.9	36
20	10	2.0	22	55	63	30.7	98	90	90	67.4	71	125	98	88.3	36
21	11	2.2	25	56	64	31.9	98	91	90	68.2	69	126	98	88.7	35
22	13	2.6	29	57	65	33.0	99	92	91	69.0	68	127	99	89.1	34
23	14	2.9	33	58	66	34.1	99	93	91	69.8	67	128	99	89.5	34
24	16	3.3	37	59	67	35.3	99	94	92	70.5	65	129	99	89.9	33
25	17	3.8	41	60	69	36.4	99	95	92	71.3	64	130	100	90.3	32
26	18	4.3	46	61	70	37.6	99	96	92	72.0	63	131	100	90.7	31
27	20	4.8	49	62	70	38.7	99	97	92	72.7	62	132	100	91.0	30
28	22	5.4	54	63	72	39.9	100	98	93	73.4	61	133	100	91.4	30
29	25	6.1	56	64	73	41.0	99	99	93	74.1	61	134	100	91.7	29
30	26	6.7	60	65	74	42.1	99	100	93	74.8	60	135	100	92.0	28
31	28	7.4	63	66	75	43.3	99	101	94	75.5	59	136	100	92.3	27
32	30	8.2	65	67	76	44.4	99	102	94	76.2	59	137	100	92.7	27
33	32	8.9	67	68	77	45.6	100	103	95	76.9	58	138	100	93.0	25
34	33	9.7	69	69	77	46.7	100	104	95	77.5	57	139	100	93.2	25
35	35	10.5	70	70	77	47.9	99	105	95	78.2	56	140	100	93.5	24

5. Data analysis techniques

Before applying any software reliability model to a set of failure data it is advisable to determine whether the failure data does, in fact, exhibit reliability growth. If a set of failure data does not exhibit increasing reliability as testing progresses, there is no point in attempting to estimate and predict the system's reliability. Since the proposed models are failure count models, the test may only be applied to data in which the test intervals are of equal length. Therefore, we divided the time interval (0,t] into k units of time of equal length. The two trend tests that are commonly carried out are [10]:

- *Arithmetic mean test*. This test consists of computing the arithmetic mean τ_k of the observed times n_i, i=1,2,…,k.

$$\tau_k = \frac{1}{k}\sum_{i=1}^{k} n_i \,, \tag{18}$$

An increasing sequence of τ_k indicates reliability growth and a decreasing sequence indicates reliability decay.

• *Laplace Test*. This test is superior from an optimality point of view and is recommended for use when the NHPP assumption is made. In terms of n_i, the number of failures during unit of time i, the expression of the Laplace factor is:

$$u_k = \frac{\sum_{i=1}^{k}(i-1)n_i - \frac{k-1}{2}\sum_{i=1}^{k} n_i}{\sqrt{\frac{k^2-1}{2}\sum_{i=1}^{k} n_i}}, \tag{19}$$

In practice, in the context of reliability growth, negative values indicate a decreasing failure intensity and thus a reliability increase, positive values suggest an increasing failure intensity and thus a reliability decrease, and values oscillating between −2 and +2 indicate stable reliability.

In other words, in order to determine whether the software underwent a reliability growth or not, we apply both the arithmetic mean and Laplace trend test to the failure data.

6. Model validation and comparison criteria

The performance of a software reliability model judged by its ability to fit the past software failure data (i.e., goodness of fit criteria) and to predict satisfactorily the future behavior from present and past data behavior (i.e., predictive validity criterion).

We evaluate the performance of the models under comparison given in Table 1 using MSE, Bias, Variation, and RMSPE metrics. The smaller the metric value the better [15, 24].

• *The mean square fitting error (MSE)*. The models under comparison are used to simulate the failure data, the difference between the expected values, $\hat{m}_{t_i}$ and the observed data x_i is measured by MSE as follows:

$$MSE = \frac{1}{k}\sum_{i=1}^{k}\left(\hat{m}_{t_i} - x_i\right)^2 \tag{20}$$

where k is the number of observations.

• *Bias*. The difference between the observation and prediction of number of failures at any instant of time i is known as PE_i (prediction error). The average of PE_s is known as bias:

$$Bias = \frac{1}{k}\sum_{i=1}^{k} PE_i \tag{21}$$

where PE_i=*Actual*(*observed*)$_i$−*Predicted*(*estimated*)$_i$,

• *Variation*. The standard deviation of prediction error is known as variation:

$$Variation = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}\left(PE_i - Bias\right)^2} \tag{22}$$

• *Root Mean Square Prediction Error* (*RMSPE*). It is a measure of closeness with which a model predicts the observation:

$$RMSPE = \sqrt{\left(Bias^2 + Variation^2\right)}$$

Predictive validity is defined as the capability of the software reliability model to determine the future failure behavior from present and past failure behaviour. This capability is significant only when failure behavior is changing [20].

Assume that we have observed x_k failures by the end of operating time t_k. We use the failure data up to time t_e ($\leq t_k$) to estimate the parameters of $\hat{m}$. Substituting the estimates of the parameters in the mean value function yields the estimate of the number of failures $\hat{m}_{t_i}$ by time t_k. The estimate is compared with the actually observed number x_k. This procedure is repeated for various values of t_e. We can visually check the predictive validity by plotting the relative error against the normalized time:

$$Relative\ Error = \frac{\hat{m}_{t_k} - x_k}{x_k}, \text{ and } Normalized\ Time = \frac{t_e}{t_k} \tag{24}$$

The error will approach zero as t_e approaches t_k. If the points are positive (negative), the model tends to overestimate (underestimate) the future failure phenomenon. Numbers closer to zero imply more accurate prediction. The relative error is said to be acceptable if it is within ±10 percent [12].

7. Filed softwarereliability data analyses and model comparisons

7.1. Trend analysis

Figures 1 and 2 trace the arithmetic mean and Laplace trend tests respectively. Both trend tests indicate reliability decay which is expected and considered normal at the start of a new activity, such as a new life cycle phase, changing test sets within the same phase, adding new users, activating the system with a different user profile, or may also result from regression defects. Since the decay last for short period we should not pay attention to it. The reliability decay followed by reliability growth is usually welcome because it indicates that, after removal of the first defect, the corresponding activity reveals fewer and fewer defects [10]. Since the failure data exhibits increasing reliability as testing progresses, there is a point in attempting to estimate and predict the system's reliability. As both trend tests show reliability decay followed by reliability growth, which suggest the use of S-shaped models. Therefore, the proposed models with S-shaped mean value functions can be applied to failure data displaying a trend that behaves according to their assumptions.

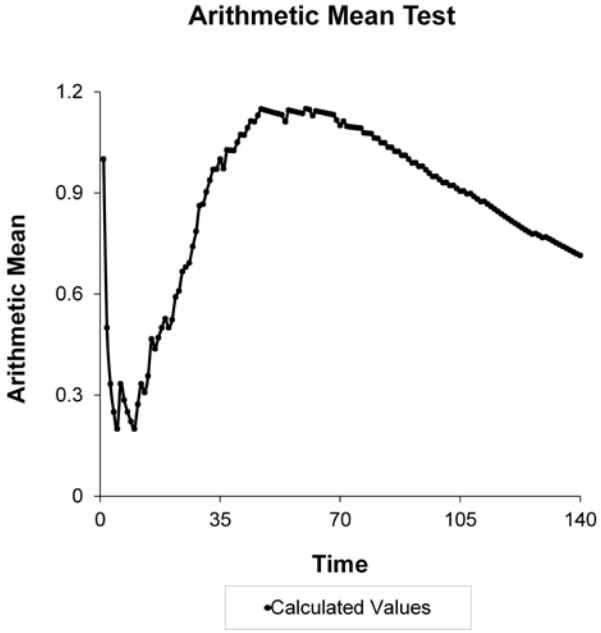

Fig. 1. Arithmetic mean trend test

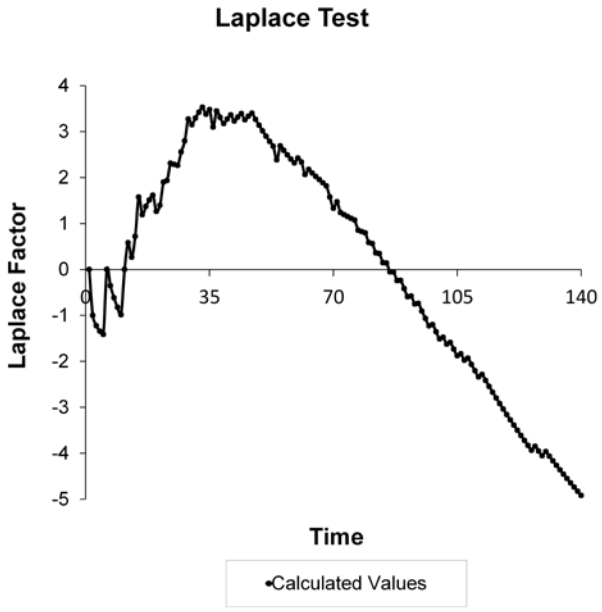

Fig. 2. Laplace trend test

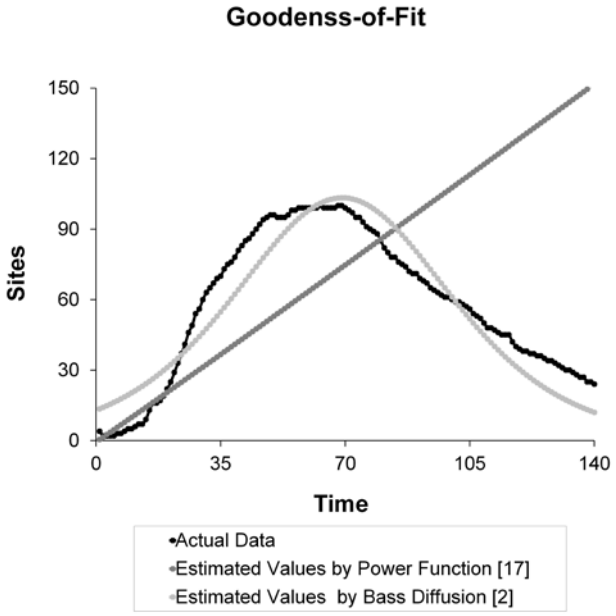

Fig. 4. Non-cumulative usage curves

7.2. Goodness of fit criteria analysis

The resultant parameters estimation and the performance of the usage functions under comparison are tabulated in Table 3. From this table, we can see that the innovation diffusion model [2] has lower MSE, bias, variation, and RMSPE metric values. Therefore, the innovation diffusion model is good enough to give a more accurate description of software usage in the operational phase.

Table 3. Parameter estimation and comparison criteria metrics results

Usage Functions Under Comparison	Parameter Estimation				Comparison Criteria			
	κ	τ	μ	η	*MSE*	*Bias*	*Variation*	*RMSPE*
Power Function [17]	1.02	—	—	—	1168418	−455	984	1084
Innovation Diffusion [2]	—	8134.5	.00163	.0475	18422	34	132	136

— the parameter is not part of the corresponding function

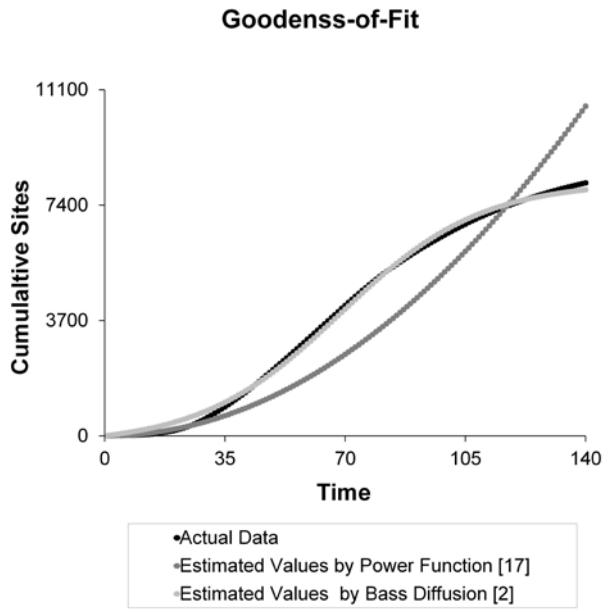

Fig. 3. Cumulative usage curves

The fitting of the usage functions under comparison to the actual cumulative and noncumulative usage are graphically illustrated in Figures 3 and 4 respectively. The power function [17] shows a poor fitting while innovation diffusion model [2] fits the data excellently as seen in Figure 3. It is clearly seen from Figure 4 that the number of sites/users who adopted this particular software product is increasing at a rapid rate and there is a stability after which it decreases in the presence of competitors or other reasons (e.g., next release becomes available). From these figures, we can observe that the innovation diffusion model [2] provides more accurate description of usage than the power function [17].

The parameter estimation and comparison criteria results of the models under comparison can be viewed in Table 4. The parameters of the innovation diffusion [2] were estimated by the least squares estimation method and given in Table 3. Using these estimated values, the MLE method is then applied to estimate the remaining parameters of the models under comparison. If we look at the estimation results, we notice that the value of parameter 'β', i.e., is greater than zero, which implies the S-shaped nature defined by the fault detection mean value function for this model. Besides, the value of parameter 'α', is zero, which implies the debugging process is perfect. It is worth noting that the second proposed model given in Eq. (13) reduces to the model [13] when applied due to its built-in flexibility. In addition, we can see that both of the proposed models provide improved results because of lower MSE, bias, variation, and RMSPE metric values.

The fitting of the models under comparison to the actual filed data are graphically illustratedin Figures 5 and 6. From Figure 5 we notice the behavior of failure data and observe that it is S-shaped in nature. This further justified by the use of the proposed models to detect the defects in the software. It is clearly seen from Figure 6 that the evolution of the failure intensity is not monotonous decreasing but S-shaped, i.e., first increasing-then-decreasing. Failure intensity has been proven to be very useful for allocating resources and determining when to stop testing in commercial systems. The distribution of failure occurrence during operation as depicted in Figure 6 shows the number of failure occurrence during an interval has a higher rate in the initial stages, reaches a maximum number per interval and then

Table 4. Parameter estimation and goodness of fit metric results

Models under Comparison	Parameters Estimation					Comparison Criteria			
	a	*b*	*γ*	*α*	*β*	MSE	Bias	Variation	RMSPE
Model due to [27]	105.37	.005617	.061249	—	—	3.85	.287	1.95	1.97
Model due to [13]	103.65	.020765	.017874	—	3.969	3.06	−.007	1.76	1.76
Proposed in Eq.(10)	103.31	.035260	.010778	—	—	3.42	−.004	1.85	1.85
Proposed in Eq.(13)	103.65	.020765	.017874	0.0	3.969	3.06	−.007	1.76	1.76

— indicates the parameter is not part of the corresponding model

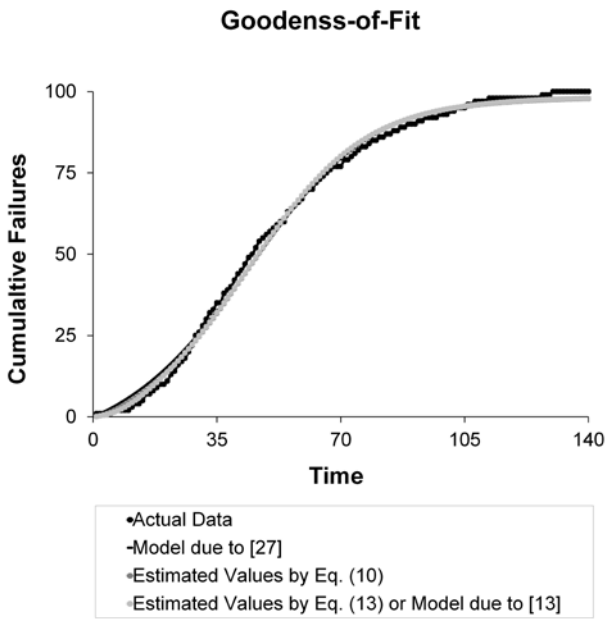

Fig. 5. Cumulative failure removal curves

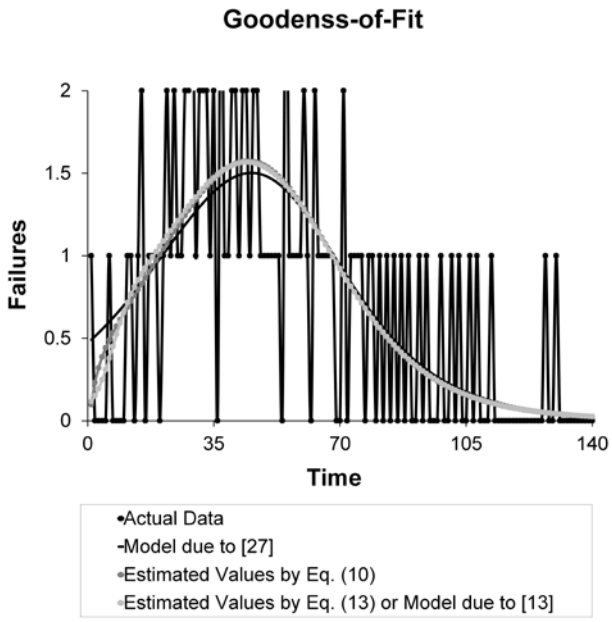

Fig. 6. Non-Cumulative failure removal curves

exponentially reduces over time toward zero. In other words, we may conclude that as the cumulative failure count increases, the failure intensity decreases.

7.3. Predictive Validity Analysis

The filed data is truncated into different proportions and used to estimate the parameters of the proposed models. For each truncation, one relative defect is obtained. Figure 7 graphically illustrates

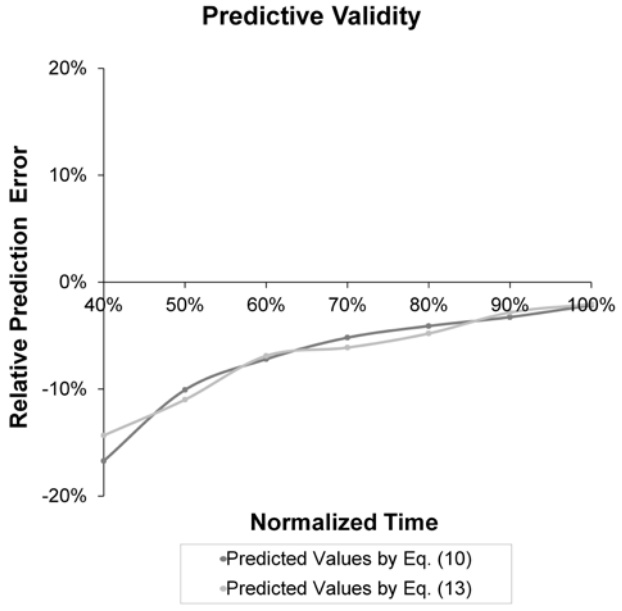

Fig. 7. Predictive validity

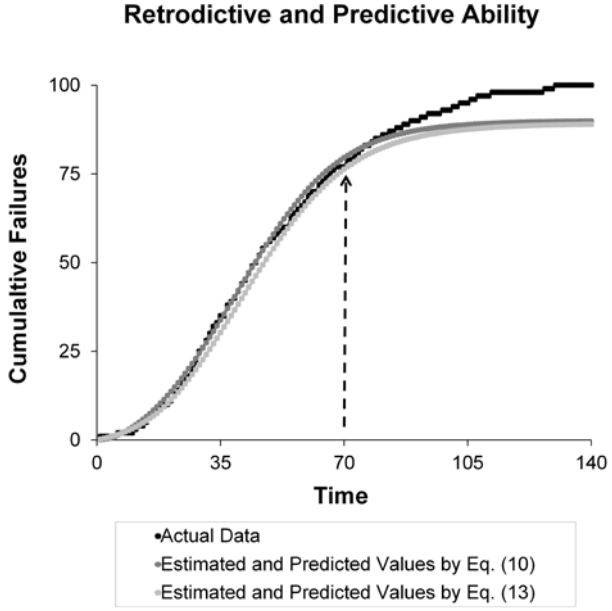

Fig. 8. Retrodictive and predictive ability

the result of the predictive validity. It is observed that the predictive validity of the model varies from one truncation to another. The error relative of the proposed models underestimates the failure observation process. Figure 8 graphically illustrates the retrodictive and predictive ability of the proposed models. The data is truncated at te (50% approx.) to estimate the proposed model parameters. The proposed models are then used to estimate the whole data. The points before t_e (marked by a dotted arrow) demonstrate the retrodictive ability while the points afterte demonstrate the predictive ability of the proposed models. It is clearly seen that 50% of the normalized time is sufficient to predict the future behavior of the failure process reasonably, which enable proper planning and of the maintenance effort. This, in turn, minimizes the maintenance cost without decreasing customer satisfaction.

8. Concluding remarks

Software reliability model is a mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment. NHPP based Software reliability models have been quite successful tools in practical software reliability engineering. These models consider the debugging process as a counting process characterized by its mean value functions. Software reliability, can be estimated once the mean value function is determined. Model parameters are usually estimated using either the maximum likelihood method or least squared estimate. They have been widely used to estimate the reliability of software during testing. Many authors have even tried to extend them to represent the failure phenomenon during the operational phase, typically used in release time problem of software. But this approach is not correct when usage of software is different from that during testing, which is actually the case for most of the commercial software. Commercial software brings many benefits to society, and plays a vital role in the development and maintenance of a diverse and vibrant information and communication technology sector. A commercial software developer endeavors to make its software product popular in the market by selling more and more copies of its product. Apart from satisfying customers by meeting all their requirements and attaching additional features, the developer at the same time makes constant efforts to build the software defect free. For measuring the operational reliability of a commercial software product, the main issue is the availability of software filed reliability data that is needed for determining reliability. Software development companies like Microsoft employees customer experience improvement program (CEIP) technology, to record both failure data and usage data. Since CEIP is available to a user by subscription only, the total population size of the observed group is known.

In this paper, an attempt has been made to model the software reliability growth linking it to the number of users who use a particular software release. Because the number of instructions executed depends on the number of users. The number of users is estimated through an innovation diffusion model of marketing. Once the estimated value is known, the rate at which instructions are executed can be found. The intensity with which failures would be reported depends upon this value. The software reliability models developed in the literature can now be used to model the fault exposure phenomenon. Following this the proposed models can help software companies like Microsoft to improve the quality, reliability and performance of its commercial software products. The proposed models have been evaluated by how good they can fit the filed data and how predictive they are. The results obtained from the proposed models discussed in this paper are quite encouraging, as can be viewed through the numerical illustrations shown in the tables and figures obtained after we performed the estimation on real filed reliability data sets. The numerical example concludes that the consideration of the effect of learning with two types of imperfect debugging in software reliability growth modeling assumptions can improve the descriptive performance of the models and the predictive performance as well.

There is a rise of interest in increasing interdisciplinary studies. It is essential to be able to predict the future scenario more accurately. We feel this study is an important step in that direction. The emphasis of the study is to show how one field of activity can enrich the other and vice-versa. Further studies are needed to examine the performance of the proposed models more by using many other reported filed data. Finally, we believe that the approach followed in this paper will help to a great extent and provides a large scope for further extension and generalization.

Acknowledgment:

Much of the research that has found its way into this manuscript was carried out during my sabbatical leave of 2013-2014. I am truly grateful to Al al-Bayt University for its support of my work and to the reviewers for their constructive comments.

References

1. Ahmad N, Khan MG, Rafi LS. A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging. International Journal of Quality & Reliability Management 2010; 27: 89-110.
2. Bass F. A new product growth model for consumer durables. Journal Marketing 1969; 15: 215-227.
3. Bardhan AK. Modelling in Software Reliability and its Interdisciplinary Nature. PhD thesis, New Delhi: University of Delhi, 2002.
4. Chiu K-C, Huang Y-S, Lee, T-Z. A study of Software reliability growth from the perspective of learning effects. Reliability Engineering and Systems Safety 2008; 93: 1410-1421.
5. Edris K, Shatnawi O. The Pham Nordmann Zhang (PNZ) software reliability model revisited. Proc. of the Tenth IASTED International Conference on Software Engineering, Innsbruck, Austria,15-17 February2011: 205-212.
6. Givon M, Mahajan V, Muller E. Software piracy: estimation of lost sales and the impact on software diffusion. Journal Marketing 1995; 59: 29-37.
7. Goel AL, Okumoto K Time dependent error detection rate model forsoftware reliability and other performance measures. IEEE Trans. Reliability 1979: 28 206-211.
8. Jones WD, Vouk MA. Field data analysis. in: Handbook of Software Reliability Engineering, Lyu M. (ed.), McGraw Hill, 1996.
9. Kan SH. Metrics and Models in Software Quality Engineering, Second Edition. USA: Addison-Wesley Professional, 2002.
10. Kanoun K, Kaaniche M, Laprie J-C. Qualitative and quantitative reliability assessment. IEEE Software 1997; 14: 77-87.
11. Kapur PK, Garg RB. A software reliability growth model for an error removal phenomenon. Software Engineering Journal 1992; 7: 291-294.
12. Kapur PK, Garg RB, Kumar S. Contributions to Hardware and SoftwareReliability.Singapore: World Scientific, 1999.

13. Kapur PK, Jha PC, Goswami DN, Shatnawi O, Bardhan AK. General framework for modeling software reliability growth during testing and operational use. Recent Developments in Quality, Reliability and Information Technology 2003:187-197
14. Kapur PK, Singh O, Shatnawi O, Gupta A. A discrete NHPP model for software reliability growth with imperfect fault debugging and fault generation. International Journal of Performability Engineering 2006; 2: 351-368.
15. Kapur PK, Pham H, Anand S, Yadav K. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Transactions on Reliability 2011; 60: 331-340.
16. Kapur PK, Singh O, Mittal R. Software reliability growth and innovation diffusion models: an interface. International Journal of Reliability, Quality and Safety Engineering 2004; 11: 339-364.
17. Kenny GQ. Estimating defects in commercial software during operational use. IEEE Transactions on Reliability 1993; 42: 107-115.
18. Kuo SY, Huan CY, Lyu MR. Framework for modelling software reliability using various testing-effort and fault-detection rates. IEEE Transactions on Reliability 2011; 50: 310-320.
19. Lyu M. (Ed.). Handbook of Software Reliability Engineering, New York: McGraw-Hill. 1996
20. Musa JD, Iannino A, Okumoto K. Software Reliability: Measurement Prediction Application. McGraw-Hill, 1987.
21. Ohba M. Software reliability analysis models. IBM Journal of Research and Development 1984; 28: 428-443.
22. Ohba M, Chou XM. Does imperfect debugging effect software reliability growth. Proceedings of 11th International Conference of Software Engineering, Pittsburgh, PA, USA, 15-18 May 1989: 237-244.
23. Pham H. Software reliability. London: Springer, 2000.
24. Pillai K, Nair VSS. A model for software development effort and cost estimation. IEEE Transactions on Software Engineering 1997; 23: 485–497.
25. Shatnawi O. Testing-effort dependent software reliability model for distributed systems. International Journal of Distributed Systems and Technologies 2013; 4: 1-14.
26. Shatnawi O. Discrete time NHPP models for software reliability growth phenomenon. International Arab Journal of Information Technology 2009; 6: 124-131.
27. Shatnawi O. Measuring software-operational reliability: an interdisciplinary modelling approa ch.Proc. of the IFIP 18th World Computer Congress - Student Forum, Toulouse, France, 22-27 August 2004: 165-176.
28. Trachtenberg M. A general theory of software reliability modeling. IEEE Transactions on Reliability 1990; 39: 92-96.
29. Wang S, Wu Y, Lu M, Li H. Discrete nonhomogeneous Poisson process software reliability growth Models based on test coverage. Quality Reliability Engineering International 2013; 29: 103–112.
30. Xie M. Software reliability modelling. Singapore: World Scientific, 1991
31. Yamada S, Ohtera H, Narihisa H. Software reliability growth models with testing-effort. IEEE Transactions on Reliability 1986; R-35, 19-23.
32. Yamada S, Ohba M, Osaki S. S-shaped reliability growth modelling for software error detection. IEEE Transactions on Reliability 1983; 32: 475-484.
33. Yamada S, Tokuno K, Osaki S. Imperfect debugging models with fault introduction rate for software reliability assessment. Int'l Journal of System Science 1992; 23: 2253-64.
34. Zhang X, Teng X, Pham H. Considering fault removal efficiency in software reliability assessment. IEEE Transactions on Systems, Man and Cybernetics 2003; 33: 114-120.

Omar SHATNAWI
Department of Computer Science
Al al-Bayt University
Mafraq 25113, Jordan
E-mail: dromali@lycos.com