Hindawi
Mathematical Problems in Engineering
Volume 2017, Article ID 8040346, 6 pages
https://doi.org/10.1155/2017/8040346

Research Article

Machine Learning Approach for Software Reliability Growth Modeling with Infinite Testing Effort Function

Subburaj Ramasamy and Indhurani Lakshmanan

School of Computing, SRM University, Kattankulathur, Tamil Nadu 603203, India

Correspondence should be addressed to Indhurani Lakshmanan; indhurani.a@gmail.com

Received 2 February 2017; Revised 22 June 2017; Accepted 27 June 2017; Published 26 July 2017

Academic Editor: Erik Cuevas

Reliability is one of the quantifiable software quality attributes. Software Reliability Growth Models (SRGMs) are used to assess the reliability achieved at different times of testing. Traditional time-based SRGMs may not be accurate enough in all situations where test effort varies with time. To overcome this lacuna, test effort was used instead of time in SRGMs. In the past, finite test effort functions were proposed, which may not be realistic as, at infinite testing time, test effort will be infinite. Hence in this paper, we propose an infinite test effort function in conjunction with a classical Nonhomogeneous Poisson Process (NHPP) model. We use Artificial Neural Network (ANN) for training the proposed model with software failure data. Here it is possible to get a large set of weights for the same model to describe the past failure data equally well. We use machine learning approach to select the appropriate set of weights for the model which will describe both the past and the future data well. We compare the performance of the proposed model with existing model using practical software failure data sets. The proposed log-power TEF based SRGM describes all types of failure data equally well and also improves the accuracy of parameter estimation more than existing TEF and can be used for software release time determination as well.

1. Introduction

Early Software Reliability Growth Models (SRGMs) represent the relationship between the time to failure and the cumulative number of faults detected till then. Many such SRGMs have been proposed as parametric [1–14] and nonparametric [15–18] models since the year 1972 to estimate future failure occurrence times and assess the reliability growth of software systems during the testing phase. The traditional SRGMs are based on the premise that the mean value function of the model follows either exponential growth [1, 3] or S-shaped growth [2, 11] or both [4–8].

Some SRGMs have been proposed with testing effort function (TEF) [11–13], since the fault detection and correction depend on efforts consumed such as test cases executed, man-days expended, computer utilization time, and other resources consumed, rather than only testing time or calendar time. The effort based SRGMs proposed in the past use exponential, Rayleigh, logistic, or Weibull distributions to specify testing effort function (TEF) to denote effort consumption during testing [11–13]. Although these functions seem to give good result and can well fit in some cases, there is a fallacy in assuming finite total test effort at an infinite time. Xie and Zhao proposed a Nonhomogeneous Poisson Process (NHPP) reliability growth model based on log-power distribution which is a graphical model where fitting of the data or not can be visualized in a graph before parameter estimation [19].

In this paper, we propose using log-power [19] distribution to describe TEF in Goel and Okumoto [1] SRGM to provide an SRGM with infinite TEF. We use Artificial Neural Network (ANN) for parameter estimation and apply machine learning technique to determine the most suitable weights for the proposed model that will fit the past and future data equally well. We study and compare the goodness of fit (GoF) performance of the proposed model with a popular test effort function based SRGM. We use ANN for parameter estimation uniformly in all cases since ANN improves the parameter estimation accuracy and gives better goodness of fit rather than traditional statistical parametric models [15–18].

Table 1: Testing effort functions.

Models	Equation	Notations
Weibull TEF [11]	$W(t) = \alpha\left(1 - e^{(-\beta t^{\gamma})}\right)$	$W(t)$: cumulative testing effort consumption in $(0, t]$ α: expected total testing effort consumed during testing β: scale parameter and γ: shape parameter
Logistic TEF [12]	$W(t) = \dfrac{N}{1 + Ae^{(-\alpha t)}}$	N: total testing effort consumed A: constant α: consumption rate of testing effort expenditure
Proposed log-power TEF	$W(t) = n\ln^{c}(1 + t)$	n and c are constants

This paper is organized in the following manner. Section 2 presents the proposed testing effort function. Section 3 presents the proposed Software Reliability Growth Model. Section 4 gives the approach to check the validity of the proposed model. Section 5 describes parameter estimation using ANN. Section 6 presents the machine learning technique used to select appropriate weights of the proposed model. Section 7 describes the performance analysis. Section 8 describes one application of the proposed model, namely, software release time determination. Summary and conclusions are given in Section 9.

2. Proposed Testing Effort Function

Since the resources consumed during software testing directly impact software reliability improvement, few SRGMs with testing effort functions were proposed in the past. To study the properties of testing effort functions, we compare the proposed log-power TEF with already proposed test effort functions such as Weibull and logistic. The comparison is given in Table 1.

The exponential and Rayleigh TEFs are special cases of Weibull TEF when the shape parameter is 1 and 2, respectively. Weibull TEF displays a peak curve when the shape parameter in the Weibull function increases. The exponential TEF is used, when the effort is uniformly consumed on the testing time whereas the Rayleigh TEF is used when the testing effort first increases to a peak and then decreases. In case of logistic TEF, at time "$t = 0$," the effort $W(t) = W(0)$ is nonzero. It is unrealistic, because at the initial stages when time is zero no testing effort can be consumed.

There are innumerable chances for faults creeping in software systems. Therefore one has to adopt a strategy for the generation of effective test cases for minimizing the error content. It is believed that achieving zero defect in software is possible but impractical due to the requirement of infinite efforts. At time "$t = 0$," the effort $W(t) = W(0)$ is not zero since test cases and test plan are drawn before testing starts. Thereafter it grows with testing. We chose log-power TEF because of its simplicity with just two parameters and it was found to be growing logarithmically with time and representing real testing projects better.

3. Proposed Software Reliability Growth Model with Log-Power Testing Effort Function

Instead of proposing a brand new SRGM for the sake of it, we propose building on the past good work done by researchers [1, 19]. We time transform the G-O model using log-power testing effort function. In the classical Goel-Okumoto SRGM, the independent variable, that is, time "t," is replaced with log-power testing effort function "$W(t)$" by applying the time transformation as applicable to NHPP models [20].

If $W(t)$ is the log-power testing effort spent at time t then the mean value function $\mu(t)$ of the Goel-Okumoto model can be transformed as given below:

$$\begin{aligned} \mu(t) &= a\left(1 - e^{(-b*W(t))}\right), \\ W(t) &= n * \ln^{c}(1 + t), \end{aligned} \tag{1}$$

where $W(t)$ is total testing effort consumed in time interval $(0, x]$, a is the expected number of software errors to be detected, and b, c, and n are constants.

Thus, the mean value function $\mu(t)$ of the SRGM with log-power TEF is as follows:

$$\mu(t) = a\left(1 - e^{(-b(n*\ln^{c}(1+t)))}\right). \tag{2}$$

4. Checking Validity of the Model

We evaluate the performance of the proposed model by using four practical software failure data sets which are available in the form of (t_n, w_n, y_n). The data set needs to be normalized in the range of $[0, 1]$ before feeding to the ANNs. Table 2 provides the description of the software failure data sets.

We measure and compare the goodness of fit (GoF) performance of the proposed model by using Mean Square Error (MSE) [22]. MSE is used to measure the square of the difference between the actual and estimated values. The smaller MSE indicates the less fitting error and better performance.

Step 1

(1) Normalize the input and output data set patterns.

(2) Initialize the weight values to small random numbers.

(3) Set the error rate condition criteria.

(4) Derive the activation functions for hidden layer and output layer from the mean value function.

Step 2

Calculate the Input value and Output value for hidden and output neurons using activation functions.

(1) Output of Input Neuron j = The input data value p_i

(2) The output of Hidden neurons or output neuron $p_j = A(\text{NET})$ where $\text{NET} = \sum_i w_{ji} \cdot p_i$, A is the activation function, w_{ji} is the weight from i to j and p_i is the input data set pattern values.

Step 3

Calculate error ∂, Start from output layer & work backward to hidden layers recursively.

Error $\partial_o = (d - o) * (A^1(\text{NET}))$ where ∂_o is the error for output layer neurons.

Similarly calculate error ∂_j for hidden layer neurons.

Step 4

Do the Weight Adjustments for output and hidden layers.

Weight adjustments for output layer $w_{ji} = w_{ji} + \Delta w_{ji}$ where $w_{ji} = c \cdot \partial_o \cdot p_i$ here c is the learning rate co-efficient and the weights are adjusted by gradient descent method in which the weight change is proportional to the partial derivative of the error.

Repeat step 2 to step 4 until the stopping criteria are met.

Box 1: ANN feed-forward back-propagation procedure.

Table 2: Software failure data sets.

Data sets	Description
DS-1 [20]	Release-1 which is a subset of four software releases cited from Wood for Tandem Computers Company. It was tested for 20 weeks (t_n) in which 100 software failures (y_n) were found and 10000 CPU hours (w_n) were consumed.
DS-2 [20]	Release-2 from Tandem Computers Company. It was tested for 19 weeks (t_n) in which 120 software failures (y_n) were found and 10272 CPU hours (w_n) were consumed.
DS-3 [20]	Release-3 from Tandem Computers Company. It was tested for 12 weeks (t_n) in which 61 software failures (y_n) were found and 5053 CPU hours (w_n) were consumed.
DS-4 [21]	Cited from Brooks and Motley which was tested for 35 weeks (t_n) and 1301 failures (y_n) were found & 1846.92 CPU hours (w_n) were consumed.

5. Parameter Estimation Using Artificial Neural Network

We use feed-forward ANN with back-propagation algorithm for estimating parameters of the proposed model. Thus, the mean value function of the proposed SRGM with log-power TEF (2) is given as follows:

$$Y(t) = w_4\left(1 - e^{(-w_3(w_2 * \ln^{w_1}(1+t)))}\right), \tag{3}$$

where w_1, w_2, w_3, and w_4 are the weights of software reliability model and their values are determined using ANN. Here, the activation functions of the ANN are developed according to the mean value function of the selected SRGM and testing effort function [16]. In order to estimate the weight values, software failure data which is available in the form of (t_n, w_n, y_n) is used where t_n is the cumulative testing time which is measured in terms of appropriate time such as months and hours, w_n is the effort expended in terms of number of hours, and y_n is the corresponding cumulative number of failures.

First, we estimate w_1 and w_2 values for log-power TEF $W(t) = w_2 * \ln^{w_1}(1 + t)$ using software failure data pair (t_n, w_n). Then, w_3 and w_4 values are estimated for mean value function $Y(t) = w_4(1 - e^{(-w_3(W(t)))})$ using software failure data pair (w_n, y_n) and here w_n is the estimated values of $W(t)$. The activation functions for hidden layers are $\ln(1+t)$ and $1 - e^{(-t)}$. The linear activation function t is used for output layers.

The ANN feed-forward back-propagation procedure for parameter estimation is given in Box 1.

6. Machine Learning Technique to Select Appropriate Weights of the Proposed Model

The goodness of fit statistic indicates the quality of fitting of past data. The objective is not only to get a better fit for the past data, but also to ensure that the model will describe the future data equally well. Traditionally the predictive validity, both short-term and long-term of the software reliability models, was measured in order to confirm that the model will describe the future data well. We apply hold-out cross-validation approach which is one of the conventional machines' learning technique to get the better goodness of fit for the past data as well as predictive validity to describe the future data [23].

Multiple sets of weights may lead to equally good fit when we use ANN. Different good fits are possible depending on the start values assigned at random for the weights. Selection of weights based only on minimum training error could be

(1) Train on the 60% training data set.
(2) Calculate the training data set accuracy by propagating error through the network and by adjusting the weights using ANN feed-forward back-propagation algorithm.
(3) Validate on the next 20% validation data set.
(4) If
the threshold validation accuracy is met stop training.
Else
continue training until the threshold validation accuracy met.
(5) Test on the next 20% test data set to confirm the selection of the model with appropriate weights.

Box 2: Description of machine learning cross-validation procedure to select appropriate weight values.

Table 3: MSE for training and validation.

Data sets	Weight sets	Training MSE	Validation MSE
DS-1	Trial-1	0.4829	1.6251
	Trial-2	**0.4831**	**0.4829**
DS-2	Trial-1	0.4823	1.5382
	Trial-2	**0.4742**	**0.4745**
DS-3	Trial-1	0.3465	1.3074
	Trial-2	**0.3482**	**0.3478**
DS-4	Trial-1	0.4157	1.3876
	Trial-2	**0.4028**	**0.4031**

misleading since the model may not describe future data accurately in the same manner. If the selected weights result in low training error but have high validation error, it is due to high variance or overfitting. Hence after arriving minimum training error (for 60% training data set) with the selected weights, we carry out validation (for 20% nonoverlapping validation data set) to ensure that the model will fit new data adequately. Box 2 describes the cross-validation procedure to select appropriate weights of the model.

Table 3 provides the Mean Squared Error values for both training and cross-validation for two trial weight sets of the proposed model.

It can be seen that although training error is more or less the same for both Trial-1 and Trial-2, the validation error is significantly higher for Trial-1 for both data sets. So it will not describe the future data better. Since the training and validation errors are both lower for the Trial-2 weights, the model will fit the future data also equally well.

7. Performance Analysis

Once the appropriate weights of the proposed model are determined as above, then the model is tested for performance using the remaining 20% test data to confirm the selected weights. The MSE calculated with test data is given in Table 4.

To study the relative performance of the testing effort function, we compare the proposed log-power TEF with already proposed Weibull test effort function [11], both used in G-O model [1]. The results confirm the suitability of log-power test effort function which appears to be the logical choice for TEF.

Table 4: MSE for test data.

Models	MSE			
	DS-1	DS-2	DS-3	DS-4
G-O SRGM with Weibull TEF [11]	0.5327	0.5262	0.4028	0.8935
Proposed G-O SRGM with log-power TEF	0.4823	0.4741	0.3485	0.4031

8. Determining When to Stop Testing: Use of Proposed SRGM

When to stop testing and release the software for operational use is one of the applications of Software Reliability Growth Models [22, 24]. Since the estimation of optimum release time based on conditional reliability does not converge [19], release time determination was carried by Subburaj and Gopal using minimum target failure intensity as the criterion instead of reliability [5], which converged after a few phases of testing. We adopt the same approach to determine when to stop testing using the proposed model. Box 3 describes the procedure for software release time determination using failure intensity to stop testing.

The equation of failure intensity function of proposed log-power TEF based SRGM is given as follows:

$$\lambda(t) = \frac{dy}{dt} = \frac{d\left(w_4 \cdot \left(1 - e^{(-w_3 * w_2 \cdot (\ln(1+t)^{w_1}))}\right)\right)}{dt},$$

$$\lambda(t) = \frac{w_4 \cdot w_3 \cdot w_2 \cdot w_1 \cdot e^{-w_3 \cdot w_2 \cdot \ln(t+1)^{w_1}} \cdot \ln(t+1)^{(w_1 - 1)}}{t+1}. \quad (4)$$

A target failure intensity of 1.663 failures per week is set for software failure data set DS-4. The target failure intensity has been achieved, and testing can be stopped at 25 weeks by which time 1166 failures were observed as given in Table 5. When we use effort based SRGM we can not only find the optimum testing time (T_{OPT}), but also determine the effort needed to achieve target reliability as illustrated in Table 5.

(1) Set the target failure intensity to stop testing that depends on the software failure data set and customer requirements.
(2) In Phase I, estimate the parameters upto 25% of total number of failures in the software failure data set.
(3) Find the Optimal Testing Time (T_{OPT}) needed to meet the target failure intensity using $\lambda(t)$.
(4) In the next Phase, estimate the parameters upto T_{OPT} with the software failure data set.
Repeat step (3) with the updated estimated parameters.
(5) Find the T_{OPT}.
if
it is less than or equal to target then stop testing.
else
repeat step (3) till the target failure intensity achieved with the required T_{OPT}.

Box 3: Procedure for software release time determination to stop testing.

Table 5: Release time determination of the proposed SRGM for DS-4.

Phase	Time (in weeks)	Number of failures	W_4	W_3	W_2	W_1	T_{OPT}	Effort needed
1	10	312	1350.04	$1.54E-03$	265.78	0.2158	17	645
2	17	771	1350.08	$1.35E-03$	650.62	0.3127	25	1200
3	25	1166	1350.15	$1.15E-03$	1185.84	0.4910	25	1200

9. Summary and Conclusions

In time-based Software Reliability Growth Models (SRGMs), we assume that the testing efforts are constant over time which may be unrealistic at times. Effort based SRGMs are more realistic and result in better goodness of fit. Hence, some SRGMs with testing effort functions were proposed in the past. We propose log-power TEF which is an infinite test effort function, since logically the test efforts will be infinite at the infinite testing time. The proposed log-power TEF based SRGM describes all types of failure data equally well. The goodness of fit indicates the quality of fitting of past data. It does not assure that the future data will be fitted equally well. Hence we determine the appropriate weights using machine learning technique to select the SRGM that will describe both the past and future failures equally well. The study confirms that SRGM with log-power TEF improves the accuracy of parameter estimation more than existing TEF and can be used for software release time determination as well. Instead of conventional parameter estimation methods, we use ANN for parameter estimation. Although already proposed SRGM uses Weibull distribution for effort function, our study reveals the log-power TEF to be simple and equally good and it is a natural choice for TEF. It is clear that the proposed log-power TEF based SRGM which is selected using machine learning technique improves the accuracy of the goodness of fit performance better than the Weibull TEF based SRGM which is already proposed.

Conflicts of Interest

The authors declare that there are no conflicts of Interest regarding the publication of this paper.

References

[1] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[2] S. Yamada and S. Osaki, "Software Reliability Growth Modeling: Models and Applications," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1431–1437, 1985.

[3] P. K. Kapur and R. B. Garg, "A software reliability growth model for an error removal phenomenon," *Software Engineering Journal*, vol. 7, no. 4, pp. 291–294, 1992.

[4] A. L. Goel, "Software reliability models: assumptions, limitations, and applicability," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1411–1423, 1985.

[5] S. Ramasamy and G. Govindasamy, "Generalized exponential Poisson model for software reliability growth," *International Journal of Performability Engineering*, vol. 2, no. 3, pp. 291–301, 2006.

[6] R. Subburaj, G. Gopal, and P. K. Kapur, "A software reliability growth model for Vital Quality Metrics," *South African Journal of Industrial Engineering*, vol. 18, no. 2, pp. 93–108, 2007.

[7] S. Ramasamy and G. Govindasamy, "A software reliability growth model addressing learning," *Journal of Applied Statistics*, vol. 35, no. 9-10, pp. 1151–1168, 2008.

[8] R. Subburaj, G. Gopal, and P. K. Kapur, "A software reliability growth model for estimating debugging and the learning indices," *International Jounal of Performability Engineering*, vol. 8, no. 5, pp. 539–549, 2012.

[9] S. Ramasamy and A. M. J. Muthu Kumaran, "Dynamically weighted combination of fault - Based Software Reliability Growth Models," *Indian Journal of Science and Technology*, vol. 9, no. 22, Article ID 93967, 2016.

[10] S. Ramasamy and C. A. S. Deiva Preetha, "Dynamically weighted combination model for describing inconsistent failure data of software projects," *Indian Journal of Science and Technology*, vol. 9, no. 35, 2016.

[11] S. Yamada, J. Hishitani, and S. Osaki, "Software-reliability growth with a Weibull test-effort: a model and application," *IEEE Transactions on Reliability*, vol. 42, no. 1, pp. 100–105, 1993.

[12] C.-Y. Huang, S.-Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 198–211, 2007.

[13] P. K. Kapur, D. N. Goswami, A. Bardhan, and O. Singh, "Flexible software reliability growth model with testing effort dependent learning process," *Applied Mathematical Modelling*, vol. 32, no. 7, pp. 1298–1307, 2008.

[14] J. Xu and S. Yao, "Software reliability growth model with partial differential equation for various debugging processes," *Mathematical Problems in Engineering*, Article ID 2476584, Art. ID 2476584, 13 pages, 2016.

[15] P. Roy, G. S. Mahapatra, P. Rani, S. K. Pandey, and K. N. Dey, "Robust feed forward and recurrent neural network based dynamic weighted combination models for software reliability prediction," *Applied Soft Computing*, vol. 22, pp. 629–637, 2014.

[16] Y.-S. Su and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *Journal of Systems and Software*, vol. 80, no. 4, pp. 606–615, 2007.

[17] I. Lakshmanan and S. Ramasamy, "An Artificial Neural-Network Approach to Software Reliability Growth Modeling," in *Proceedings of the 3rd International Conference on Recent Trends in Computing, ICRTC 2015*, pp. 695–702, March 2015.

[18] S. Ramasamy and I. Lakshmanan, "Application of artificial neural network for software reliability growth modeling with testing effort," *Indian Journal of Science and Technology*, vol. 9, no. 29, Article ID 90093, 2016.

[19] M. Xie and M. Zhao, "On some reliability growth models with simple graphical interpretations," *Microelectronics Reliability*, vol. 33, no. 2, pp. 149–167, 1993.

[20] M. Xie, *Software Reliability Modeling*, World Scientific, Singapore, Asia, 1991.

[21] A. Wood, "Predicting software reliability," *Computer*, vol. 29, no. 11, pp. 69–77, 1996.

[22] R. Subburaj, *Software Reliability Engineering*, McGraw-Hill Professional, New Delhi, India, 2015.

[23] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, pp. 40–79, 2010.

[24] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Springer-Verlag, Tokyo, Japan, 2014.