Volume 4, No. 2, Jan-Feb 2013

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

A Systematic Review of Software Reliability and Software Cost Models

Poonam Panwar*
Assistant Professor
Department of Computer Science & Engineering
Ambala College of Engineering & Applied Research
Ambala, India
rana.poonam1@gmail.com

Ankur Garg
M.Tech Research Scholar
Department of Computer Science & Engineering
Ambala College of Engineering & Applied Research
Ambala, India
ankurgarg.garg10@gmail.com

Abstract: Software reliability assessment is important to evaluate and predict the reliability and performance of software system. The models applicable to the assessment of software reliability are called Software Reliability Growth Models (SRGMs). An SRGM provides a mathematical relationship between time span of testing or using the software and the cumulative number of faults detected. It is used to assess the reliability of the software during testing and operational phases. An important class of SRGM that has been widely studied is Non Homogeneous Poisson Process (NHPP). NHPP models are useful in describing failure processes, providing trends such as reliability growth and fault-content. Different NHPP models have been developed for different applications. In this paper, we described several existing software reliability growth models based on Non Homogeneous Poisson processes (NHPPs). This paper also addresses cost estimation models and cost functions that can be used to evaluate the cost of software during its development.

Keywords: Software Engineering; Software Reliability; SRGM; NHPP; MVF.

I. INTRODUCTION

Software reliability is the probability that software will provide failure free operation in a fixed environment for a fixed interval of time. Probability of failure is the probability that the software will fail on the next selected input. Software reliability is typically measured per units of time. Software Reliability is also an important factor affecting system reliability. Software Reliability assessment is important to evaluate and predict the reliability and performance of software system. The models applicable to the assessment of software reliability are called software reliability growth models. Software reliability growth models have been discussed abundantly in the literature. SRGMS can estimate the number of initial faults, the software reliability, the failure intensity, the mean time interval between failures, etc. An important class of SRGM that has been widely studied is Non Homogeneous Poisson Process. NHPP models are useful in describing failure processes, providing trends such as reliability growth and fault content [1]. Cost estimation models are mathematical algorithms or parametric equations used to estimate the costs of a product or project. The results of the models are typically necessary to obtain approval to proceed, and are factored into business plans, budgets, and other financial planning and tracking mechanisms. The costs of developing software and software failure have entailed great expenses in a system development. Therefore, it is important to determine when to stop testing or when to release the software to the users so that the total system cost is minimized, subject to a desired reliability level and other constraints [2].

II. NON HOMOGENEOUS POISSON PROCESS MODEL

The non-homogeneous Poisson Process group of models provides an analytical framework for describing the software failure phenomenon during testing. The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time. The NHPP represents the number of failures experienced up to time t as an NHPP, $\{N(t),t\geq 0\}$ denote a counting process representing the cumulative number of faults detected by the time t. An SRGM based on an NHPP with the mean value function (MVF), m(t) can be formulated as in equation 1[3].

$$P\{N(t)=n\}=\frac{m(t)^n}{n!}e^{-m(t)},\ n=0,1,2 \qquad (1)$$

Where m(t) represents the expected cumulative number of faults detected by the time t. The MVF m(t) is non decreasing with respect to testing time t under the bounded condition $m(\infty)=a$, where a is the expected total number of faults to be eventually detected. Knowing its value can help us to determine whether the software is ready to be released to the customers and how much more testing resources are required. It can also provide an estimate of the number of failures that will eventually be encountered by the customers. Generally, we can get distinct NHPP models by using different non-decreasing mean value functions [3]. The failure intensity function at testing time t is given in equation 2.

$$\lambda(t)=\frac{dm(t)}{dt}=m'(t) \qquad (2)$$

The software reliability, i.e., the probability that no failures occur in (s, s+t) given that the last failure occurred at testing time s($s\geq 0$, $t>0$), is given below in equation 3[4].

$$R((t|s)=\exp\left[-\left(m(t+s)-m(t)\right)\right] \qquad (3)$$

The fault detection rate per fault at testing time t is given by equation 4.

$$d(t)=\frac{m'(t)}{a-m(t)}=\frac{\lambda(t)}{a-m(t)} \qquad (4)$$

There are several existing well-known NHPP models with different MVFs. Some of them are described below.

a. Nhpp Exponential Model: The exponential NHPP model is based on the following assumptions[5]:

a) All faults in a program are mutually independent from the failure detection point of view.
b) The number of failure detected at any time is proportional to the current number of faults in a program. This means that the probability of the failure for faults actually occurring i.e., detected, is constant.
c) The isolated faults are removed prior to future test occasions.
d) Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced.

$$m(t) = a[1 - e^{-bt}] \quad (5)$$

where a is the expected total number of faults that exist in the software before testing and b is the failure detection rate or the failure intensity of a fault. This model is also known as Goel-Okumoto model.

b. Musa Exponential Model: Musa proposed a similar model to the Goel-Okumoto model by considering the relationship between execution time and calendar time [6]. Let m(t) be the number of the failures discovered as a result of test case runs up to the time of observation. Mean value function obtained by Musa is given in equation 6.

$$m(t) = a\left(1 - e^{-\frac{ct}{n\tau}}\right) \quad (6)$$

c. Hyper-exponential Growth Model: The hyper exponential growth model is based on the assumption that a program has a number of clusters of modules, each having a different initial number of errors and a different failure rate. The mean value function of the hyper exponential class NHPP model is given in equation 7 [7].

$$m(t) = \sum_{i=1}^{n} a_i[1 - e^{-b_i t}] \quad (7)$$

d. Yamada And Osaki Model: A similar extension of the exponential growth model has been suggested by Yamada and Osaki by dividing software into k modules. The failure intensity of faults within different modules are assumed to be different, while the failure intensity of faults within the same module are assumed to be the same. The mean value function is given in equation 8 [8].

$$m(t) = a\sum_{i=1}^{k} p_i[1 - e^{-b_i t}] \quad (8)$$

e. Connective NHPP Model: Nakagawa (1994) proposed a model, called connective NHPP model, where the basic shape of the growth curve is exponential and that an S-curve forms due to the test. The mean value function is given in equation 9 [9].

$$m(t) = a_1\left(1 - e^{-bI_{[0.t_0)}(t)}\right) + a_2\left(1 - e^{-bI_{[t_{0.\infty})}(t)}\right) \quad (9)$$

f. NHPP S-Shaped Model: In the NHPP s-shaped model, the software reliability growth curve is an S-shaped curve which means that the curve crosses the exponential curve from below and the crossing occurs once and only once. The NHPP s-shaped model is based on the following assumptions and the mean value function is given in equation 10[10]:

a) The error detection rate differs among faults.
b) Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced.

$$m(t) = a\left[1 - e^{-\int_0^t b(u)du}\right] \quad (10)$$

g. NHPP Inflection S-Shaped Model: The inflection S-shaped model is based on the dependency of faults by postulating the following assumptions and the m(t) is given in equation 11 [11].

a) Some of the faults are not detectable before some other faults are removed.
b) The probability of failure detection at any time is proportional to the current number of detectable faults in the software.
c) Failure rate of each detectable fault is constant and identical.
d) The isolated faults can be entirely removed.

$$m(t) = \frac{a}{1 + \beta e^{-bt}}\left(1 - e^{-bt}\right) \quad (11)$$

h. NHPP Delayed S-Shaped Model: The delayed S-Shaped model is based on the following assumptions and the mean value function is given in equation 12 [12]

a) All faults in a program are mutually independent from the failure detection point of view.
b) The probability of failure detection at any time is proportional to the current number of faults in software.
c) The proportionality of failure detection is constant.
d) The initial error content of the software is a random variable.
e) A software system is subject to failure at random times caused by errors present in the system.
f) The time between failures (i-1)th and ith depends on the time to the (i-1)th failure.
g) Each time a failure occurs, the error which caused it is immediately removed and no other errors are introduced.

$$m(t) = a\left[1 - (1 + bt)e^{-bt}\right] \quad (12)$$

i. NHPP Imperfect Debugging Model. NHPP Imperfect debugging model is based on the following assumptions and mean value function is given by equation 13 [13]

a) When detected errors are removed, it is possible to introduce new errors.
b) The probability of finding an error in a program is proportional to the number of remaining errors in the program.
c) The probability of introducing a new error is constant.
d) Three type of errors exist:

Type 1 errors (critical): very difficult to detect
Type 2 errors (major): difficult to detect.
Type 3 errors (minor): easy to detect.

e) The parameters a and b_i for i=1, 2, 3 are unknown constants.
f) The error detection phenomenon in the software is modeled by an NHPP.

$$m(t) = \sum_{i=1}^{3} m_i(t) \quad (13)$$

$$\text{where } m_i(t) = \frac{ap_i}{1-\beta_i}\left[1 - e^{-(1-\beta_i)b_i t}\right] \qquad (14)$$

j. NHPP Imperfect Debugging S-Shaped Model: NHPP Imperfect debugging S-Shaped model is based on the following assumptions and the model can be formulated by the differential equation 15[14]

a) The error detection rate differs among faults.
b) Each time a software failure occurs, the software error which caused it is immediately removed, and new faults can be introduced.

$$\frac{\partial m(t)}{\partial t} = b(t)[a(t) - m(t)] \qquad (15)$$

k. Pham Exponential Imperfect Debugging Model: The model is based on the following assumptions and the mean value function is given by equation 16 [15]

a) The introduction rate is an exponential function of testing time, and
b) The error detection rate function is non-decreasing with an inflection S-shaped model.

$$m(t) = \frac{\alpha_1 b}{b+\beta}\left(\frac{e^{(\beta+b)t} - 1}{e^{bt} + c}\right) \qquad (16)$$

l. Pham-Zhang NHPP Model : This mean value function of the model is given in equation 17 and the model is based on following assumptions [16].

a) The error introduction rate is an exponential function of the testing time in other words, the number of errors increases quicker at the beginning of the testing process than at the end. This reflects the fact that more errors are introduced into the software at the beginning, while at the end, testers posses more knowledge and therefore introduce fewer errors into the program.
b) The error detection rate function is non-decreasing with an inflection S-shaped model.

$$m(t) = \frac{1}{(1+\beta e^{-bt})}\left((c+a)\left(1 - e^{-bt}\right) - \frac{ab}{b-\alpha}\left(e^{-\alpha t} - e^{-bt}\right)\right) \qquad (17)$$

III. SOFTWARE COST MODELS

In the recent years, the costs of developing software and software failure have entailed great expenses in a system development. Therefore it is important to determine when to stop testing or when to release the software to the users so that the total system cost is minimized, subject to the desired level of reliability and other constraints. In this section we present cost models and cost functions that can be used to formulate realistic total software cost projects in many applications and to determine the optimal release policies of the software system.

a. **SOFTWARE COST MODEL WITH RISK FACTOR:** this cost model addresses the risk level and the time to remove errors and the optimal release policies that minimize the expected total software cost. The expected software system cost, E (T), is defined as [17]:

a) The cost to perform testing.
b) The cost incurred in removing errors during the testing phase.
c) A risk cost due to software failure.

The cost to perform testing is given by:

$$E_1(T) = C_1 T \qquad (18)$$

The expected total time to remove all $N(T)$ errors is

$$E\left[\sum_{i=1}^{N(T)} Y_i\right] = E[N(T)]E[Y_i] = m(T)\mu_y \qquad (19)$$

Hence, the expected cost to remove a errors detected by time T can be expressed as

$$E_2(T) = C_2\, E\left[\sum_{i=1}^{N(T)} Y_i\right] = C_2 m(T)\mu_y \qquad (20)$$

The risk cost due to software failure after releasing the software is

$$E_3(T) = C_3\,[1 - R(x|T)] \qquad (21)$$

where C_3 is the cost due to software failure.

The expected total software cost can be expressed as

$$E(T) = C_1 T + C_2 m(T)\mu_y + C_3\,[1 - R(x|T) \qquad (22)$$

b. Generalized Software Cost Model: It considers the cost of removing errors detected during the warranty period and risk cost due to software failure. Additional assumptions [18]

a) There is set up cost at the beginning of the software development process.
b) The cost of testing is a power function of the testing time. This means that at the beginning of the testing, the cost increases at higher gradient, slowing down later.
c) The time to remove each error during the warranty period follows a truncated exponential distribution.
d) The cost to remove errors during the warranty period is proportional to the total error of removing all errors detected between the intervals of (T,T_w).

$$E(T) = C_0 + C_1 T^{\alpha} + C_2\, m(T)\mu_y + C_3\mu_w[m(T+T_w) - m(T)] + C_4[1 - R(X|T)] \qquad (23)$$

c. Cost Model with Multiple Failure Errors: In this section a cost model with is presented with following assumptions [19]:

a) The cost of debugging an error during the development phase is lower than in the operational phase.
b) The cost of removing a particular type of error is constant during the debugging phase.
c) The cost of removing a particular type of error is constant during the operational phase.
d) The cost of removing critical errors is more expensive than major errors, and the cost of removing major errors more expensive than minor errors.
e) There is a continuous cost incurred during the entire time of the debugging period.

The expected software cost

$$E(T) = \int_0^T \left[C_3 + \sum_{i=1}^{3} C_{i1} m_i(t)\right] g(t)dt + \int_T^{\infty} [C_3 T + \sum_{i=1}^{3} C_{i1} m_i(T) + \sum_{i=1}^{3} C_{i2}\left(m_i(t) - m_i(T)\right) g(t)dt \quad (24)$$

IV. SOFTWARE COMPONENT COST FAILURE INTENSITY FUNCTIONS

a. Linear cost function: The linear form can be used in the absence of knowledge about the relative nonlinear characteristics of cost to attain reliability, or when this relationship is thought to be indeed linear. The advantage of linear cost function is simplicity. Disadvantage is that solution might not be practical for large or complex systems. The linear cost function is given in equation 26 [20].

$$C(\lambda) = \begin{cases} -\alpha\lambda + \beta & \text{if } \lambda \le \dfrac{\beta}{\alpha} \\ 0, & \text{otherwise} \end{cases} \quad (26)$$

b. Logarithmic Exponential Cost Function: LnExp function is in nonlinear form. An advantage of the Lnexp cost function is that only a single parameter β needs to be specified for each component. A disadvantage of the LnExp function is that there is little flexibility for changing the shape of the curve. The LnExp cost function is given in equation 27 [21].

$$C(\lambda) = \begin{cases} -\beta \ln(1 - \exp(-\lambda)), & \text{if} \lambda > 0 \\ +\infty, & \text{otherwise} \end{cases} \quad (27)$$

c. Inverse Power Cost Function: Another nonlinear function that satisfies the desired characteristics is that inverse power (InvPow) cost function. The advantage of the InvPow cost function is that it is a generalization of the basic COCOMO introduced by Bohem. A disadvantage of the the InvPow cost function is that either two or three parameters per component must be specified, depending on whether the location parameter δ is assumed to be nonzero as given in equation 28 [22].

$$C(\lambda) = \begin{cases} \dfrac{\beta}{(\lambda - \delta)^{\alpha}} & \text{if} \lambda > \delta \\ +\infty & \text{otherwise} \end{cases} \quad (28)$$

V. CONCLUSION

This paper reviews software reliability papers published in journals classified according to research topic, research approach, and study context. In this paper, we described software reliability and software cost models that can be used to predict the optimal release time of software. Hence, it would help a software vendor to calculate the total software product cost and its reliability.

VI. REFERENCES

[1] B. Littlewood, "Software Reliability Modeling: Achievements and Limitations", Proc. of Fifth Ann. European Computer Conf. on Advanced Computer Technology, Reliable system, and Applications (CompEuro'91), May1991, pp. 36-344.

[2] R.H. Hou, S.Y. Kuo, and Y.P. Chang, "On a Unified Theory of Some Nonhomogenous Poisson Process Models for Software Reliability", Proc. of International Conf. on Software Eng., Education & Practice (SEEP '98), Jan. 1998, pp. 60-67.

[3] M. Trachtenberg, "A General Theory of Software-Reliability Modeling," IEEE Trans. Reliability, vol. 39, no. 1, Jan. 1990, pp. 92-96.

[4] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," IEEE Trans. Software Eng., vol. 11, no. 12, Dec. 1985.

[5] Y.K. Malaiya and P.K. Srimani, "Software Reliability Models: Theoretical Developments, Evaluation and Applications," IEEE Press, 1990.

[6] L. Goel and K. Okumoto, "A Time Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures," IEEE Trans. on Reliability, vol. 28. no. 3, 1979, pp 206—211.

[7] M. Ohba, "Software Reliability Analysis Models," IBM J. Research and Development, vol. 28, no. 4, July 1984, pp. 428-443.

[8] N. Langberg and N.D. Singpurwalla, "A Unification of Some Software Reliability Models," SIAM J. Scientific and Statistical Computing, vol. 6, no. 3, Mar. 1985, pp. 781-790.

[9] Yamada S. and Osaki S., "Discrete Software Reliability Growth Models," Applied Stochastic Models and Data Analysis, vol. 1, no.1, 1985, pp. 65-77.

[10] S. Yamada M. Ohba, and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," IEEE Trans. On Reliability, vol. R-32, no. 12, 1983, pp 475-478.

[11] Helander, M.E., Zhao,M., Ohlsson, N. "PlanningModels for Software Reliability and Cost," IEEE Trans. in Software Engineering, vol. 24, no. 6, June 1998.

[12] W. Wang, Y. Wu, and M.H Chen,"An Architecture –Based Software Reliability Model," Proc. Pacific Rim Dependability Symp., 1999.

[13] S.Gokhale And K.S Trivedi, "Time/Structure Based Software Reliability Model," Annals of Software Eng., vol.8, 1999, pp.85-121.

[14] S.S Gokhale, T.Philip, P.N.Marinos, and K.S Trivedi,"Unification of Finite Failure Non-Homogeneous Poisson Process Models Through Test Coverage," Proc. Seventh Int'l Symp. on Software Reliability Eng.,1996.

[15] H. Pham, "Software Reliability," Springer-Verlag, 2000.

[16] Hui Guan, Wei-Ru Chen, Ning Huang and Hong-Ji Yang "Estimation Of Reliability And Cost Relationship For

Architecture Based Software," International Journal of Automation and Computing, November 2010.

[17] J. D. Musa, and K. Okumoto, "Software Reliability Measurement, Prediction, Application," McGraw Hill, 1987.

[18] M.R. Lyu, "Handbook of Software Reliability Engineering," McGraw-Hill, 1996.

[19] A. D. Denton, "Accurate Software Reliability Estimation," Master of Science Thesis, Colorado State University, Fort Collins, Colorado, Fall 1999.

[20] Q. P. Hu,M. Xie,S.H. Ng,and G. Levitin," Robust recurrent neutral network modeling for Software Fault Detection and Correction Prediction," Reliability Engineering and System Safety, vol 92 no.3,2007,pp. 332-340.

[21] D. R. Jeske, and X. Zhang,"Some successful approaches to software reliability modeling in industry," J. Syst. Softw., vol. 74, no. 1, 2005, pp.85-99.

[22] C. Y. Huang, and C. T. Lin, "Software reliability analysis by considering fault dependency and debugging time lag," IEEE Trans. Reliability, vol.55, no. 3, 2006, pp. 436–450.