Progress in Nonlinear Dynamics and Chaos
Vol. 2, No. 1, 2014, 10-17
ISSN: 2321 – 9238 *(online)*
Published on 27 July 2014
www.researchmathsci.org

Progress in
Nonlinear
Dynamics and Chaos

Software Reliability Modeling in Fuzzy Environment

***M. Kiruthiga*[1] and *C. Loganathan*[2]**

Department of Mathematics, Maharaja Arts and Science College
Coimbatore, Tamilnadu
[1]Email: Kiruthi.neha@gmail.com; [2]Email: clogu@rediffmail.com

Received 13 July 2014; accepted 26 July 2014

Abstract. The idea of software reliabilities can be represented by the quantity of defects in the unit time. The association between the intensity of the defects and reliability depends upon the model that is used in the evaluation. Many models try to assess whether a software testing objective has been met to determine when to stop testing. We present a case study for evaluating software reliability on the OFBiz project software package used on business routines. We also explain a new model that can have more accurate analysis and forecast to software assumptions. Fuzzy theory is introduced to evaluate software reliability.

Keywords: Software Reliability, Nonlinear Models, Fuzzy Logic, Business Routines.

1. Introduction

Due to the complexity of computer software systems and the increasing of development cost, it is of utmost importance to develop high quality software systems. The quality of the software system is described by many metrics such as: complexity, maintainability, availability, reliability, etc. As tragedies of unreliable software often take place, people recognize the importance of developing reliable software. Now, the reliability problem in software systems is a well-known research field. Therefore, designing reliable software and evaluation software reliability accurately are the most important issues [1].

The standard definition of reliability for software (Musa, Iannino, and Okomoto, 1987) is the probability of execution without failure for some specified interval of natural units or time [2]. Software failures are the manifestation of software errors which are introduced in to the software by software engineers during the phases of software development cycle. In the literatures, researchers establish software reliability growth model (SRGM) to measure software reliability. To apply these models, it is necessary to know how well the models suit an actual observation failure data set. In order to obtain accurate software reliability estimation, it requires a large number of failure data which are not usually available until the system has been tested for a long relative period. Many software reliability engineers are more interested in estimating the software reliability as early as possible [3].

2. Growth models

Software reliability is the probability that software will not cause the failure of a product for specified time under specified conditions, this probability is a function of the inputs to

and use of the product, as well as a function of the existence of faults in the software, the inputs to the product will determine whether an existing of faults in the software, the inputs to the product will determine whether an existing fault is encountered or not. Many software reliability growth models have been developed over the years. For a detailed description of most models consider Musa, Iannino, and Okumoto. Within these models one can distinguish two main categories: Predictive models, assessment models. Predictive models typically address the reliability of the software early in the life cycle at the requirements or at the preliminary design level or even at the detailed design level in a waterfall life cycle process or in the first spiral of a spiral software development process. Predictive models could be used to assess the risk of developing software under a given set of requirements and for specified personnel before the project truly starts.

- **Assessment models** - evaluate present and project future software reliability from failure data gathered when the integration of the software starts. Predictive software reliability models are few in number, most models can be categorized in the assessment category.
- **Analytical models –** analytical modeling of software reliability involves four steps. The first step is to define the assumptions associated with a software test procedure, and the second step is to develop an analytical model based on the assumptions and the test procedure. The third step is to obtain parameters for the model using the data the final step is to use the model for performance predictions. Two major types of analytical models can be identified. They are dynamic and statistic models. In a dynamics software reliability model, the time dependent behaviors of the software failures are captured in the analytic model. However, in a statistic model, no reference is made to the time dependent behavior of software failures.
- **Empirical models –** in an *empirical* software reliability model, a relationship or a set of relationships between software reliability measures and appropriately define software metrics are developed using empirical results available from past data. This model can then be applied to measure software reliability for which we have the required software metrics. This, in a way is similar to the econometric models in forecasting theory. The major issues in this modeling technique are the identification of the appropriate software metrics and the reliability measures. The latter issue of developing the functional relationship is referred to as specification. The accurate and most appropriate specification of an empirical model is a key step in the use of this technique for software reliability estimation.
- **Multi- Stage Models -** One of the assumptions made by all the models is that the set of code being testing is unchanged throughout the test period. Clearly, defect repair invalidates that assumption, but it is assumed that the effects of defect repair are minimal so that the model is still a good approximation. If a significant amount of new code is added during the test period, there is a technique that allows us to translate the data to account for the increased code change. Theoretically, the problem is that adding a significant amount of changed code should increase the defect detection rate. Therefore, the overall curve will look something like Figure 3.5, where D_1 defects are T_1 time prior to the addition of the new code and an additional $D_2 - D_1$ defects are found in $T_2 - T_1$ time after that code addition. The problem is to translate the data to a model $\mu(t)$ that

would have been obtained if the new code had been part of the software at the beginning of the test. Let $\mu_1(t)$ model the defect data prior to the addition of the new code, and let $\mu_2(t)$ model the defect data after that code addition. The model $\mu(t)$ is created by appropriately modifying the failure times from $\mu_1(t)$ and $\mu_2(t)$. This section describes how to perform the translation assuming $\mu(t)$, $\mu_1(t)$ and $\mu_2(t)$ are all G- O models. In theory, this technique could be applied to any of the models in Table 2-1, including the S- shaped models.

- **Software Growth Models:**

SRGM is defined as the mathematical relationship between the number of software errors removed and testing time. Classical SRGMs have great influence on software reliability modeling research. In spite of the fact that many software reliability growth models have been proposed in the software reliability literature since the first one appeared in 1972 [4], no one of them can deal properly with all possible situations.

According to the software error removal process, Jelinski and Moranda proposed the first model. The model has a simple structure and assumptions. Then Musa proposed the basic Execution Time Model which has similar assumptions to those of Jelinski and Moranda(J-M) model [5]. J-M Model made a major contribution to the understanding the relation of error removal and software execution time. Goel-Okumoto model is the first nonhomgeneous Poisson process (NHPP) SGRM. Goel-Okumoto model assumed that the error removal process follows NHPP. The assumption of this model is similar to the Basic Execution Time model of Musa [6].

Those models are mainly based on some assumption conditions and believe that SGRM has been established according to a certain probability process. The assumptions are the key factors of establishing SGRM. There is a relation between assumption chosen and modeling success. But in partial application, quite a few assumptions are unfit to software developing process, which cannot be accepted by most people, thus limiting the application of the models. Using this kind of model can even get the unimaginable parameter under some conditions. Furthermore, a software failure data set being evaluated can get different results by using different models. So it has reached a common viewpoint in software reliability evaluation field, in that there isn't a "good model" that can fit all failures data very well [7].

J-M model assumes that all errors have been checked in the test can be eliminated and removes one error each time. Removing one error does not affect the remaining errors. But in fact, software development and error removal are all human behaviors, which are unpredictable, so it cannot avoid of introducing new errors during the process of error removal. Goel-Okumoto model suppose that each error is independent, each error has the same probability of leading system failure and each failure interval time is independent. Sometimes, modules in software program have some relations, so it is impossible to have complete independence, and the probability of each failure that causes system failure is not the same [8].

In these three models, the cumulative numbers of errors removal grow exponentially with the testing time. The exponential growth curve is due to the assumption that the error removal intensity is linearly related to the remaining number of software errors [9]. In many software development projects it was observed that the relation between the cumulative numbers of errors removed and the testing time is not linear. So we should establish nonlinear model to evaluate the software reliability.

3. The feasibility of software reliability nonlinear modeling based on time series

Time series analysis theory is a method of describing statistics character of dynamics data, which can set up time series model from limited sample data, its advantage is convenience and practicality. There are many literature on the development of estimation and prediction in auto regression time series models. Time series analysis method is well studied in some statistical literatures. However, its use in software reliability engineering is rather limited [10].

Time series prediction can be stated as follows: given a finite sequence X_1, X_2, X_3,…,X_i, predicting the continued sequence X_{i+1}, X_{i+1}, …. For example, $\{X_i\}$ can be viewed as the stochastic failure intervals or the number of failures per time intervals mainly. That is to say, software reliability failure data are discrete data sequence, whether it is steady or not, we can use the data to modeling and evaluate software reliability by applying proper time series method [11].

During the process of software reliability evaluation, it can be seen that the nonlinear phenomena exist very commonly and we cannot deal these data with linear model. The relevant software reliability time series nonlinear models are deuced in the following section. This modeling method proves that software reliability evaluation also can be presented in the way of time series nonlinear model.

4. The implemented algorithms

The cumulative number of failures *M(k)* is increasing and trend to a fixed value which is defined as the desired cumulative number of failures [12].

Assume cumulative failures of software are stochastic variable. We can find that all total failures in one time interval are exponential decreasing according to experience. Then we have software reliability nonlinear model

$$M(k) = b^k M(k-1) + M(k-1). \tag{1}$$

Let $\theta(k) = b^k$, then the software reliability nonlinear model can be transformed to time series model as follow:

$$M(k) = (\theta(k)+1)M(k-1) + \varepsilon(k), \tag{2}$$

where $\varepsilon(k)$ is zero-average white noise. Obviously, we can get

$$\theta(k) = b\theta(k-1). \tag{3}$$

That's to say, parameter $\theta(k)$ can be described by the following AR time series model

$$\theta(k) = b\theta(k-1) + \xi(k) \tag{4}$$

where $\xi(k)$ is zero-average white noise.

As testing time is based on the unit of day, the cumulative numbers of software failures fluctuate greatly. Considering the series $\{\theta(k)\}$ got from observed testing data exist must disturbance noise, we apply a smoothing filter to testing data as following steps: first, we get series $\{\eta(k)\}$ simply as following equation

$$\eta(k) = \frac{M(k)}{M(k-1)} - 1,\ k = 2, \dots, N \tag{5}$$

and let $\eta(1) = \eta(2)$; then we can get time series $\{\theta(k)\}$ from $\{\eta(k)\}$ by applying smoothing filter:

$$\theta(k) = \lambda\theta(k-1) + (1-\lambda)\eta(k), \tag{6}$$

where the initial is $\theta(1) = \eta(1)$ and $\lambda = 0.7$ in the simulation.

Now we can use exponential weighted Least Squares method [13] to estimate the parameter b from equation (4). Formulas can be shown as the following equations:

$$\hat{b}(k+1) = \hat{b}(k) + (k+1)[\theta(k+1) - \hat{b}(k)\theta(k)] \tag{7}$$

$$K(k+1) = \frac{P(k)\theta(k)}{\omega + \theta^2(k)P(k)} \tag{8}$$

$$P(k+1) = \frac{1}{\omega}[1 - K(k+1)\theta(k)]P(k) \tag{9}$$

where ω is forgetting factor, $0 < \omega < 1$, commonly is about 0.9~0.99. The initial values are given by:

$$\hat{b}(0) = 1,\ P(0) = 10^4.$$

Then we can calculate the estimation as follows:

$$\hat{\theta}(k) = \hat{b}(k-1)\theta(k-1). \tag{10}$$

$$\hat{M}(k \mid k-1) = (\hat{\theta}(k)+1)M(k-1) \tag{11}$$

Based on these results, we can get the prediction from the following function:

$$\hat{\theta}(N+p \mid N) = \hat{b}(N)\theta(N+p-1 \mid N),\quad p = 1,2,\dots, \tag{12}$$

$$\hat{M}(N+p \mid N) = (\hat{\theta}(N+p \mid N)+1)\hat{M}(N+p-1 \mid N), \tag{13}$$

$$\textit{where}\quad \hat{\theta}(N \mid N) = \hat{\theta}(N),\ \hat{M}(N \mid N) = M(N).$$

5. Autoregressive models

Stationary process and time series

The series of observations x(t), t∈T made sequentially in time t constitutes a time series. Examples of data taken over a period of time are found in abundance in diverse fields such as meteorology, geophysics, biophysics, economics, commerce, communication engineering systems analysis, etc. Daily records of rainfall data, prices of a commodity etc. constitute time series. The variate t denotes time, i.e., changes occur in time, but this need not always be so [14].

For example, the records of measurements of the diameter of a nylon fiber along its length (distance) t also give a time series. Here t denotes length.

Autoregressive process (AR process)

The process $\{X_t\}$ given by

$X_t + b_1 X_{t-1} + b_2 X_{t-2} + \text{-------} + b_h X_{t-h} = e_t$, $b_h \neq 0$,

where $\{e_t\}$ is a purely random process, with mean 0, is called an autoregressive process of order h.

Autoregressive process of order two (yule process)

This is given by

$X_t + b_1 X_{t-1} + b_2 X_{t-2} = e_t$,

where $\{e_t\}$ is a purely random process, with mean 0.

Day	Cf
0	7
3	12
5	15
13	26
29	38
32	50
40	90
46	175
50	200
55	240
60	250
65	275
70	325
75	390
80	450
85	470
90	500

$y_5 + b_1 y_4 + b_2 y_3 + b_3 y_2 + b_4 y_1 = 0$ (14)

$y_6 + b_1 y_5 + b_2 y_4 + b_3 y_3 + b_4 y_2 = 0$ (15)

$y_7 + b_1 y_6 + b_2 y_5 + b_3 y_4 + b_4 y_3 = 0$ (16)

$y_8 + b_1 y_7 + b_2 y_6 + b_3 y_5 + b_4 y_4 = 0$ (17)

$y_9 + b_1 y_8 + b_2 y_7 + b_3 y_6 + b_4 y_5 = 0$ (18)

By solving the above equations,

b1 = -0.5749

b2 = 1.0021609

b3 = -0.000352

b4 = -504402

6. Software reliability and its fuzzy evaluations

When $\theta(k)$ is closer to zero, the number of remaining errors is less and software reliability is higher. So the objective of software reliability can be designed as $R(k) = -\log(\hat{\theta}(k))$.

Strictly speaking *R(k)* is a fuzzy variable we can compartmentalize some grades to *R(k)* and use fuzzy variable and fuzzy rule to estimate software reliability.

7. Conclusion and future enhancement

We have fitted a fuzzy model and regressive model. Second model is found to be better than fuzzy model.

REFERENCES

1. Nunes Rodrigues, Genaina, A Model Driven Approach for Software Reliability Prediction, University of London, doctorial Thesis, 2008.
2. Vittorio Gianni Fougastsaro, A Study of Open Source ERP System, Thesis for Masters Degree in Business Administration, 2009.
3. Michael R. Lyu, Software Reliability Engineering: A Roadmap, Future of Software Engineering (FOSE'07), Washington, DC, USA, IEEE, Computer Society, pp. 153 – 170, 2007.
4. A.Avizienis, J C Laprie and B. Randell, Fundamental Concepts of Dependability", in IARP/IEEE-RAS Workshop on Robot Depedability, 2001.
5. J.D.Musa and K.Okumoto, A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, ICSE 84 Proceedings of the 7th International Conference on Software Engineering, pp. 230-238, 1984.
6. H.Pham, System Software Reliability, Spriger Series in Reliability Engineering, 1st Edition, 2007.
7. Z.Jelinski and P.B.Moranda, Software Reliability Research, in Statistical Computer Performance Evaluation, W. Freiberger, Ed., New York: Academic Press, pp. 465 – 484, 1972.
8. Martin L Shooman, Probabilistic Models for Software Reliability Prediction, in Statistical Computer Performance Evaluation, W. Freiberger, Ed., New York: Academic Press, pp. 485 – 502, 1972.
9. Michael R.Lyu, Allen P.Nikora, William H. Harr, A Systematic and Comprehensive Tool for Software Reliability Modeling and Measurement, Fault-Tolerant Computing. FtCS-23. Digest of Papers, The Twenty-Third International Symposium, pp. 648 – 653. 1993.
10. John D.Musa, Software Reliability – Engineered Testing, *Journa Computer archive*, 29 (11) (1996) 61 – 68.
11. Z.Krajcuskova, Software Reliability Models, 17th International Conference radioelektronika, pp. 1 – 4, 2007.
12. S.Parthasarathy, N.Anbazhagan, Significance of Software Metrices in ERP Projects, India Conference Annual IEEE, pp. 1 – 4, 2006.

13. C.Freericks, Open Source Standards on Software Process: A Practical Approach, IEEE Communications Magazine, 39(4) (2001) 116 – 123.
14. J.Medhi, Stochastic Processes, New Age International Publishers, New Delhi, second edition, pp. 340 – 346, 2006.