**A study of software reliability growth from the perspective of learning effects**

**Master of Technology**

**in**

**Computer Science**



**By**

**Basrah Yousuf**

**Under the supervision of**

**Dr.Javed Iqbal**

**PG. DEPARTMENT OF COMPUTER SCIENCE**

**UNIVERSITY OF KASHMIR, SRINAGAR**

**Table 0f Contents**

**Chapter 1 INTRODUCTION**

**Chapter 2 NHPP MODELS**

**Chapter 3 PROPOSED MODEL**

**Chapter 4 DATA ANALYSIS AND MODEL COMPARISON**

**Chapter 5 CONCLUSIONS**

**Introduction**

Software reliability is defined as the probability that the system will perform its intended function under specified working conditions for a specified period of time. Software reliability growth models are mathematical functions that describe the error detection and the removal process. They help developers to know whether or not the code is suitable for use and how much more testing is required if it is not ready yet to release the software. It also provides the estimate of the number of faults that the users will encounter when operating the software. In order to balance between system stability and software testing debugging costs, software reliability growth models are widely used by software developers and testing staff. In fact, accurately modeling the software reliability and predicting its possible trends are essential in determining the optimal policy of system release. In this section, we apply an NHPP(non homogenous Poisson process) to describe the time-dependent behavior of the cumulative number of errors detected up to a certain testing time. The general model with NHPP applied is formulated based on the following assumptions:

1. The software errors testing/debugging process follows an NHPP.

2. The software fault intensity rate at anytime is proportional to the number of remaining faults in the software at that time.

3. A debugging procedure takes place immediately when a software error is detected.

Generally, the software testing/debugging process is modeled as an error counting process. A counting process {N(t),t ≥0} is said to be an NHPP with intensity function where N(t) follows a Poisson distribution with mean function m(t); The mean value function m(t) is the expected number of errors detected within time (0, t). The conditional software reliability R(x/t) is defined as the probability that no error is detected within the time interval (t, t+x), given that an error occurred at time t (t≥ 0, x>0). Note that x is an operating time period according to some practical and managerial requirements.

Note that one of the assumptions built into an NHPP is that the numbers of events occurring in disjoint time intervals are statistically independent of each other, but this assumption is not true for detection of a finite number of errors. Obviously, the events in disjoint time intervals are dependent when the number of remaining errors has become small. Therefore, the NHPP model is inappropriate when nearly all of the errors have already been found. Nevertheless, the NHPP is a widely used model in describing software reliability growth behavior. In this study, the analysis concentrates on data from the early and middle stages of the error-discovery process, so the NHPP should be an adequate model which would have better prediction accuracy for the early and middle stages than for the late stage.

**Problem Statement**

Reliability growth models have been studied to predict software reliability in the testing/debugging phase. Most of the models developed were based on the non-homogeneous Poisson process (NHPP), and S-shaped type or exponential-shaped type of behavior is usually assumed. Unfortunately, such models may be suitable only for particular software failure data, thus narrowing the scope of applications. Therefore, from the perspective of learning effects that can influence the process of software reliability growth, we considered that efficiency in testing/debugging concerned not only the ability of the testing staff but also the learning effect that comes from inspecting the testing/debugging codes. The proposed approach can reasonably describe the S-shaped and exponential-shaped types of behaviors simultaneously, and the results in the experiment show good fit. A comparative analysis to evaluate the effectiveness for the proposed model and other software failure models was also performed.

**NHPP Models**

**Goel-Okumoto Model**

This model, first proposed by Goel and Okumoto, is one of the most popular NHPP model in the field of software reliability modeling. It is also called the exponential NHPP model. Considering failure detection as a Non homogeneous Poisson process with an exponentially decaying rate function, the mean value function is hypothesized in this model as

m(t)=a[ 1- exp(-bt)]; a>0,b>0

where a and b are positive constants.

a= expected number of faults

b= failure occurrence rate.

**Yamada Delayed S-Shaped Model**

The Yamada Delayed S-Shaped model is a modification of the non homogeneous Poisson process to obtain an S-shaped curve for the cumulative number of failures detected such that the failure rate initially increases and later exponentially) decays . It can be thought of as a generalized exponential model with failure rate first increasing and then decreasing. The software error detection process described by such an S-shaped curve can be regarded as a learning process because the testers skills will gradually improve as time progresses. The mean value function is

m(t)= a (1-(1+bt)exp(-bt)); b>0

where a and b are the expected total number of faults to be

detected eventually and failure occurrence rate respectively.

**Inflected S-Shaped Model (0hba)**

This model solves a technical problem in the Goel-Okumoto model. It was proposed by Ohba and its underlying concept is that the observed software reliability growth becomes S-shaped if faults in a program are mutually dependent, i.e., some faults are not detectable before some others are removed. The mean value function is

m(t)=a((1-exp(-bt))/((1+c exp(-bt)))); a>0,b>0,c>0

The parameter c is the inflection rate that indicates the ratio of the number of detectable faults to the total number of faults in the software, a is the expected total number of faults to be eventually detected, b is the fault detection rate, and is the inflection factor.

**Yamada and others**

In general it is considered to be unrealistic in software reliability modeling to assume that the faults detected software testing are perfectly removed without introducing new faults. Yamada et.al proposed software reliability assessment models with imperfect debugging by assuming that new faults are sometimes introduced when the faults originally latent in the software system are corrected and removed during the testing phase. It is assumed that the fault detection rate is proportional to the sum of the numbers of faults remaining originally in the system and faults introduced by imperfect debugging. This model is described by a non homogeneous Poisson process. The mean value function is given by

m(t) =((ab)/(alpha+b))(exp(-alpha t)-exp(-bt)); a,b,alpha >0

**pham zang model**

m(t)=(1/(1+betaexp(-bt)))((a+c)(1-exp(-bt))-((ab)/(b-alpha))(exp(-alpha t)-exp(-bt)));a,b,c,alpha,beta>0

where alpha and beta are positive constants.

**Proposed model**

In the past, many researchers believed that the learning effect existed in the process of software testing/debugging, but they failed to identify it. Therefore, how the learning presents in a software testing/debugging task can be identified and manipulated by the manager, and also how to measure the effects of this learning, have become critical tasks in modeling software reliability.

In the proposed model, the influential factors considered for finding errors in software include the autonomous errors-detected factor alpha and the learning factor neta. Suppose that f(t) is the intensity function that denotes the fraction of the errors detected at time t, F(t) is the cumulative function that denotes the fraction of the errors detected within time (0, t), and 1-F(t) is the fraction of the errors as yet undetected at time t. the interrelationships among the factors can be simplified as a differential equation, which is given by;

f(t) = (alpha + neta\*F(t))(1- F(t))

where alpha >0 and neta >0 to specify that a constructive debugging activity is in process. It implies that imperfect debugging and learning effects exist. The autonomous errors-detected factor indicates that the testing staff/software developers spontaneously find software errors of which they were unaware. Meanwhile, the learning factor indicates that the testing staff/software developers deliberately set out to find software errors from the patterns which were previously detected. Both factors can improve the efficiency of software debugging. Since f(t) can be derived from above equation by using the differential equation analysis to solve the following equation:

f (t)=d/dt (F(t))2= neta F(t)-(alpha-neta)F(t)+alpha

the explicit solution of F(t) is given by;

F(t)=[e(alpha+neta)(t+c) (alpha/neta)]/1+e(alpha+neta)(t+c)

Note that the number of system errors found should be zero when a system’s debugging task begins, but F(t) is not equal to zero when t =0. In view of this, we utilized the constant c to adjust the function F(t), so as to let F(0) = 0.

Based on the above discussion, the constant c can be inferred as;

c = ln{(alpha/neta)}/(alpha+neta)

By substituting the constant c into above equations;

F(t) and f(t) can be rearranged as

F(t) = 1-(1+(neta/alpha)/{(neta/alpha)+e(alpha+neta)t}

and

f(t)={(alpha+neta2)e(alpha+neta)t}/{alpha((neta/alpha)+e(alpha+neta)t)2}

Note that the values of alpha and neta must be positive to ensure that the software errors would decrease gradually through the debugging process. Furthermore, we need to estimate total errors in the software system. Here, a represents the expected number of all potential errors, which includes the initially presented errors and the errors introduced by future debugging work, and which can be estimated by the size of the software system along with experience of previous testing/debugging tasks. Accordingly, the mean value function of the software error-detection process can be written as m(t)=aF(t).

* m(t)=m(t)=a(1-(1+(neta/alpha))/((neta/alpha)+(exp((alpha+neta) t)))).

Way to distinguish the exponential-shaped and the S-shaped types of behaviors is whether the value of the inflection point is greater than zero or not. The proposed model behaves as S-shaped if neta>alpha (the value of an inflection point is positive); otherwise it behaves as exponential shaped. The existence of an inflection point implies that the learning effect is relatively higher than the autonomous errors-detected rate in the process of testing/debugging.

Note that, in some special cases, the learning factor neta may not be constant, instead it varies in time. The similar analytical approach proposed in this study can still be applied but in a more complicated way by substituting neta with neta(t). Here, we deal with the constant case, and retain the issue of time-varying learning to the future work.

**Data analysis and model comparison**

we evaluate the effectiveness of the proposed model by using the different datasets from and compare the proposed model with the other models, the Goel and Okumoto model ,Yamada delayed S-shaped model, Ohba inflection S-shaped model , Yamada exponential imperfect debugging model , and Pham–Zhang model .

In order to investigate the effectiveness of the proposed model the comparison criteria chosen is R square measure.

R square (Rsq) can measure how successful the fit is in explaining the variation of the data.

First of all, we examine whether the proposed model can fit the data with different patterns. Figure 1 and 2 show the fitting results of differences between the predicted value and the actual data of cumulative errors.

Dataset 1 (Failure data of wireless data service system)



fig 1. fitting results

Dataset 2 (failure data of tandem software)

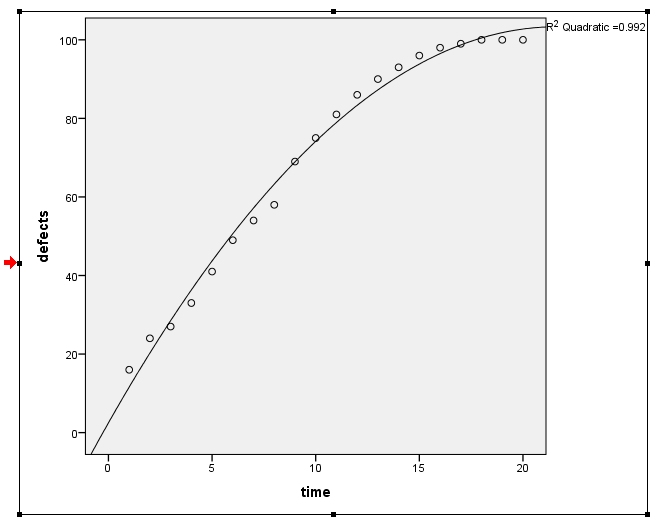


fig 2. fitting results

**Results obtained**

dataset 1

Go model : 0.986

yamada delayed S shaped : 0.986

ohba: 0.994

yamada and others: 0.990

pham and zang : 0.994

propsed model :**0.994**

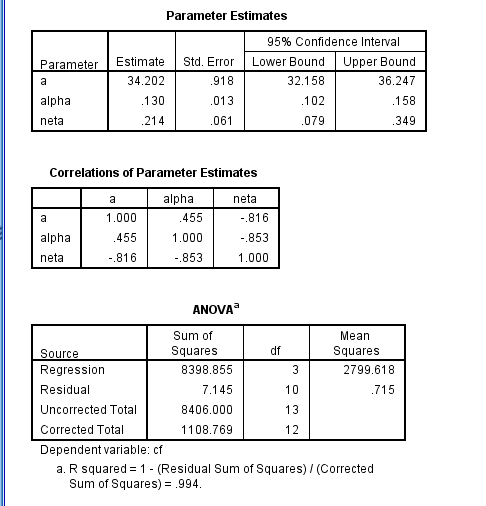


Fig 1.1 Rsq of proposed model

dataset 2

**r**Go model : 0.986

yamada delayed S shaped : 0.969

ohba: 0.989

yamada and others: 0.988

pham and zang : 0.992

propsed model :**0.989**

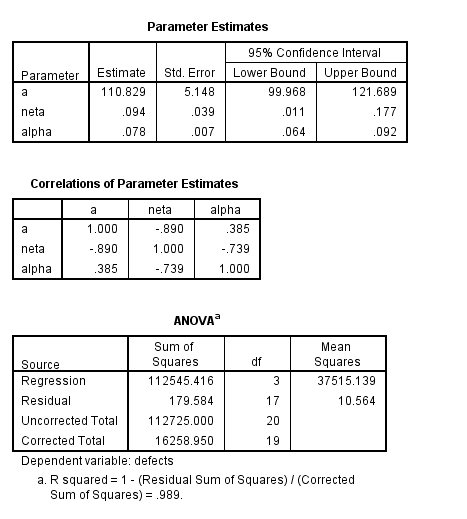


Fig 2.1 Rsq of proposed model

As can be seen in the above figures, the fitting results were fairly good for the proposed model . It indicates that the efficiency of debugging arises in the middle to later stage of the software test. Note that pham zang model performs better in the Rsq measure, but it does not consider the impact of how many parameters are utilized. In general, the more parameters contained in a model, the more flexibility it has for a data pattern, but the less reliable for its parameter estimates.

**Conclusions and future work**

In the past, only a few of the models developed could model both S-shaped and exponential-shaped types of behaviors simultaneously and identify the learning effect in the process of software reliability growth. However, the proposed model from the perspective of learning effects can reasonably describe the process of software testing/debugging. By inspecting various data and models, the proposed model not only has a goodness-of-fit but also offers a good explanation of the process of software reliability growth. It would be practicable to extend by dealing with dynamic learning effects, which means that the learning factor neta may vary in time. In such a case, an adequate function of the learning factor has to be investigated first. The other possible way of extending this project may refine the proposed model by considering the change-point problem. The change-point problem results when some factors about debugging/testing are changed, which can cause the software failure intensity function to increase or decrease.

**References**

[1] Ohba M. Software reliability analysis models. IBM J Res Dev 1984;28:428–43.

[2] Gokhale SS, Trivedi KS. A time/structure based software reliability model. Ann Software Eng 1999;8:85–121.

[3] Goel AL, Okumoto K. Time-dependent fault detection rate model for software and other performance measures. IEEE Trans Reliab1979;28:206–11.

[4] Yamada S, Ohba H. S-shaped software reliability modeling for software error detection. IEEE Trans Reliab 1983;32:475–84.

[5] Ohba M. Inflexion S-shaped software reliability growth models. In:Osaki S, Hatoyama Y, editors. Stochastic models in reliability theory.Berlin, Germany: Springer; 1984. p. 144–62.

[6] Zhao M. Change-point problems in software and hardware reliability. Commun Stat Theory Methods 1993;22(3):757–68.

[7] Shyur H-J. A stochastic software reliability model with imperfect debugging and change-point. J Syst Software 2003;66:135–41.

[8] Pham H, Zhang X. NHPP software reliability and cost models with testing coverage. Eur J Oper Res 2003;145:445–54.

[9] Huang C-Y. Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Software 2005;76:181–94.