ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 6, Issue 8, August 2017

A HYBRID METHOD FOR RELIABILITY PREDICTION OF INTERLOCKING SOFTWARE

Snehlata Gautam and Pradeep Tomar,

Department of Computer Science and Engineering,

School of Information and Communication Technology,

Gautam Buddha University, Greater Noida-201312, India,

Abstract

The major objective of this research to reduce software fault probability using Naïve bayes techniques of reliability models. That should be useful in defining which of the existing model to use in a given software development environment. In this research importance is given on comparison of existing software reliability models. The systematic models are commonly help in approximating and observing reliability. The models can help software testing/debugging managers to make predictions about the anticipated future reliability of software under growth. In existing software reliability models which claim to progress software quality through efficient determination of software faults. Failure behavior of the software as predicted by these models has important implication in understanding the performance of the software and its improvement. One unique part of this paper is that it added new algorithm Naïve byes to classify the data and predict the error of software; rather it give importance to the taxonomy of models used in the software development process. This research proposed naïve byes classification Software Reliability Models to Improve the Reliability of Software.

Keywords: Software reliability, NHPP model, Naïve byes prediction etc.

1. INTRODUCTION

In recent years, the complexity of computer programs that perform important and crucial functions is increasing rapidly, so the question of the software reliability is becoming more crucial. Using adequate models of software reliability during the design and coding phases reduce the cost of testing phase. In this paper, according to [1] reliability means the probability of failure-free operation of a software system for a specified period of time in a specified environment. Today, there are many approaches and models which can predict or assess the software reliability. In common reliability models are divided into “blackbox” models and “whitebox” models [2] depending on the usage of information about software architecture. Model is called “blackbox” when internal structure of software is unknown and conclusions can be obtained just via analysis of input/output data. In the last decade scientists pay more and more attention to “white-box” models (models which take into account information about the architecture of the software) [12, 13, 14, 15]. In turn, architecture-based models are divided into Additive models, Path-based models and State-based models.It's obvious that “whitebox” models can describe software reliability in a more adequate way, because they evaluate internal structure of a software, i.e. it's architecture.

2. Related work

In [17] The author 'Razeef Mhod & Mohsin Nazir' has summarize some existing software reliability growth models, provides critical analysis of underlying assumptions and assess the applicability of these models during the software development cycle. Software reliability growth model is a technique used to assess the reliability of the software product in

quantitative manner and this model have good performance in terms of goodness-of-fit, predictability and so forth. In [18] this paper author B. Anniprincy & S. Sridhar have Propose two dimensional Software reliability growth model by using Cobb-Douglas production function for capturing effect of testing time and testing coverage on the number of fault removed in the software system. S-shaped model is developing in the paper which we have reviewed.

In [19] The authors 'Hoang Pham, Xuemei Zhang' have proposed a model for software reliability that is incorporates with testing coverage information. This model is based on Non Homogeneous Poisson Process (NHPP) and can used to estimate and predict the software reliability of the product in a quantitative manner and we also examine the goodness-of-fit and estimation power of the model.

3. Non-Homogeneous Poisson process (NHPP)

This is a class of time domain software reliability models which assume that software failures display the behavior of a Non-Homogeneous Poisson process (NHPP). The parameter of the stochastic process, λ(t) which denotes the failure intensity of the software at time t, is time-dependent [11,12,13]. Let N(t) denote the cumulative number of faults detected at time t and m(t) denote its expectation. Then m(t) = E[N(t)] and the failure intensity λ(t) is related as follow

$$m(t) = \int_0^t \lambda(s)ds \tag{1}$$

And,

$$\frac{dm(t)}{dt} = \lambda(t) \tag{2}$$

N(t) was known to have Poisson probability density function(PDF) with parameter m(t), that is:

$$p(N(t)=n) = \frac{[m(t)]^n}{n!} e^{-m(t)}, n = 0, 1, \cdots, \infty \tag{3}$$

These time domain models for the NHPP process can be described by the probability of failure are possible. This model like as intensity function failure (failure incidence rates per fault) λ(t) conveyed differently, also mean value the function of m(t) will be expressed differently.

The NHPP models can be further classified into finite failure and infinite failure categories. Finite failure NHPP simulations accept that the predictable no.of faults noticedassumed infinite amount of testing time will be finite, whereas the infinite failures models assume that an infinite number of faults would be detected in infinite testing time [9]. Thus, using General Order Statistics (GOS) has become failure NHPP. On the other hand, using Record Value Statistics (RVS) has become infinite NHPP. Typically, the Generalized Order Statistics (GOS) model has N defects and any of the N defects from the probability density function (PDF); generated n point according to the order statistic is the time point of failure.

In this model, at the time of each repair, a new defect is assumed not to occur [10]. However, the actual situation at the point of repair new failure may occur. So as to add for this situation, the Record Value Statistics (RVS) model can be used NHPP model and mean value function was as follows: say

$$m(t) = -\ln(1 - F(t) \qquad (4)$$

Where, exp (–m(t)) = 1 – F(t) and F(t) is cumulative the distribution function and f(t) is the probability density function. Therefore, from Equation (4) using the related equations of NHPP in Equation (1), intensity function can be the hazard function (h(t)),[11]. In other words,

$$\lambda(t)=m'(t\ )f(t)\ /\ (1-F(t)=h(t) \qquad (5)$$

Let θ denote the expected number of faults that would be detected given finite failure NHPP models. Then, the mean value function of the finite failure NHPP models can also be written as:

$$m(t) = q\ F(t) \qquad (6)$$

Note that F(t) is Cumulative Distribution Function (CDF).In Equation (4), the (instantaneous) failure intensity function l(t) in case of the finite failure NHPP models is given by:

$$q(t) = \theta\ F'(t) \qquad (7)$$

This can be re-written as:

$$\lambda(t)=[\theta-m(t)]\frac{F'(t)}{1-F(t)}[\theta-m(t)]\,h(t) \qquad (8)$$

Where h(t) is the failure occurrence rate per fault of the software, or the rate at which the individual faults manifest themselves as failures course of testing process. The quantity [θ– m(t)] denotes the expected number of faults remaining in the software at time t. Since [θ – m(t)] is a monotonically non-increasing function of time (actually [θ– m(t)] should decrease as more and more faults are detected and removed1), the nature of the overall failure intensity, λ(t) is governed by the nature of failure occurrence rate per fault h(t), from Equation (6). The failure occurrence rate per fault h(t) can be a constant or increasing, decreasing.

4. GOEL-OKUMOTO GROWTH MODEL

The main objective of a software consistency model is to prediction failure performance of the software that will be qualified when the software is operational. This expected behavior changes rapidly and it can be tracked during the period in which the program is tested.

4.1 Basic Assumptions of Goel/Okumoto Model

- The execution times between the failures are exponentially distributed.
- The cumulative number of failures follows a Non Homogeneous Poisson process (NHPP) by its expected value function μ(t).
- For a period over which the software is observed the quantities of the resources that are available are constant.
- The number of faults detected in each of the respective intervals is independent of each other.
- The mean value function is such that the expected number of error occurrences for any time t to t+Δt is proportional to the expected number of undetected errors at time t. It is also assumed to be bounded, non-decreasing function of time with lim t→∞ μ(t)= N<∞

4.2 Goel-Okumoto Basic Model

$$\mu(t) = E_E\,(l\text{-}e\ {-}^{bt}),\ \text{where } E_E \geq 0,\ b>0 \tag{9}$$

$\mu(t)$ = Predicted number of defects at time t

E_E = Expected total number of defects in the code in infinite time (it is usually finite)

b = Roundness factor/shape factor = the rate at which the failure rate decreases.

t = Calendar time/ execution time/ number of test runs

Because it is a non-linear equation, the solution found may be local optimum rather global optimum. Therefore it is beneficial to define parameter values that are close to the final values [3]. The parameter values which are selected should provide a reasonable match to the existing data. But it is worthy only if the estimation is done already, or the analysis is done after the software is released to the user. If the result is obtained using the previous month's data, those parameter values are a good starting point to estimate the value of expected total no. of defects and the roundness factor.

4.3 Components of the Model

(1) Expected no. of Defects:

In this model $\mu(t)= E_EF(t)$. Here F(t) is a cumulative distribution function. F(0)=0, i.e. number of defects are 0 before the test starts, and $F(\infty)=1$, therefore $\mu(\infty)$=EE and EE is the total number of bugs detected after an infinite number if testing is done. This model attempt to statistically correlate defect detection data with other known functions like exponential functions. The models have a parameter that relates to the total number of bugs contained in the entire cod. Residual Errors [3] can be found out if the entire no. of bugs is detected and calculated as follows: Residual Errors = Total number of errors in the code - errors discovered and rectified.

(2) Roundness Factor:

The roundness factor for a perfect circle has the value "1" and for shapes with increasing irregularity, the value tends to "0". Other shape factors are sensitive especially for the presence of concave irregularities, whereas factors like the roundness factor can have the same value for shapes with many small concave irregularities and for elongated shapes without concave irregularities [4].

(3) Test Time Data:

For any software reliability growth model, the appropriate measure of time must relate to the testing effort. There are three possible methods for measuring test time:

- Calendar time
- Number of tests run
- Execution (CPU) time.

4.4. Estimation of Model Parameters:

In the case of the above model, two parameters must be estimated: total expected failures for infinite time (E_E) and the rate of reduction in the failure rate or the roundness factor (b).

The parameters can be detected during two phases:

• During the testing phase or before the software is shipped to the client. Statistical inference methods like Maximum Likelihood, Classical Least Square, and Alternative Least Square can be used to estimate the parameters in terms of calendar time.

• If the predictions are done after the software is shipped to the client, then it is done through software characteristics like size and complexity of the software or the failure data. Once the failure data is available in terms of execution time, these parameters may be estimated, using any statistical inference method [5]. The accuracy of the parameters generally increases with the size of the sample of failures.

5. Proposed Method

5.1 Naïve Bayes Model

Naïve Bayes Prediction (NBP) model chooses software module to be object unit of training and prediction. This paper used to model for improving software fault prediction module. It gives two major concerns such as: 1) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; 2) A logically separable part of a program. Now it will give the working of model.

Let A= {a_1 , a_2,…, a_n} be set of metrics attribute set, there is a vector M: {(a_1 , w_1), (a_2 , w_1),…, (a_n , a_n)} to denote a software module, where a_i is metrics and w_i is weight of a_i . It train the classifier using module data sets with category tag, and compute the defective probability of a new module which will make an alert when it exceeds a threshold.

If we define the category notion of software module is c∈{C_d, C_n}, where Cd is defective category and C_n is non-defective category. Then according to Bayesian theory [13], the probability that software module is defective will be computed by (10):

$$P[C_d \mid M] = \frac{P[M \mid C_d] \times P[C_d]}{\sum_{c \in C} P[c] \times P[M \mid c]} . \qquad (10)$$

As shown by (11), classifier will classify software module M to C_d when the ratio greater than λ,

$$\frac{P[\bar{C_d} \mid M]}{P[C_n \mid M]} > \lambda . \qquad (11)$$

P [M |c] and P[c] in (1) will be estimated using training data sets. Where P[c] will be computed by (3):

$$P[c] = \frac{N_c}{N} , \qquad (12)$$

Where N_c is training modules number of category c and N is total number of training modules. Estimation of P [M|c] is a key problem of NBP model. Some kinds of NBP models are derived from deferent estimation method of P [M|c] by researchers.

5.2 Working of Naïve Bayesian classification technique in software fault prediction

After the computation of all the metrics, testing sample data can be forwarded to the training samples to measure and classify the metric results with existing results In terms of conditional probability by using Naive Bayesian classifier, which leads to the fault prediction analysis.

5.3 Estimating probabilities

P(X), P(X|Ci),and P(Ci)may be estimated from given data Bayes Theorem:

$$P(C_i \mid X) = \frac{P(X \mid Ci)P(Ci)}{P(X)}$$

5.4 Steps Involved:

1. Each data sample is of the type

X=(xi) i =1(1)n, where xi is the values of X for attribute Ai

2. Suppose there are m classes C_i , i=1(1)m.

P(Ci |X) > P(Cj |X)

i.e. BC assigns X to class Ci having highest posterior probability conditioned on X

The class for which P(Ci |X) is maximized is called the maximum posterior hypothesis.

From Bayes Theorem

3. P(X) is constant. Only P(X|Ci), P(Ci) need be maximized.

• If class prior probabilities not known, then assume all classes to be equally likely

• Otherwise maximize P(Ci) = Si /S Problem: computing $P(X|C_i)$ is unfeasible.

4. Naïve assumption: attribute independence

$P(X|Ci) = P(x1,\ldots,xn|C_i) = \Pi P(xk|C_i)$

5. In order to classify an unknown sample X, evaluate for each class Ci. Sample X is assigned to the class $C_i ff_P$

$P(X|C_i)P(C_i) > P(X|C_j)\ P(C_j)$.

6. Results

This section presents a new model to improve the software reliability and error fault prediction using Naïve bayes prediction model.

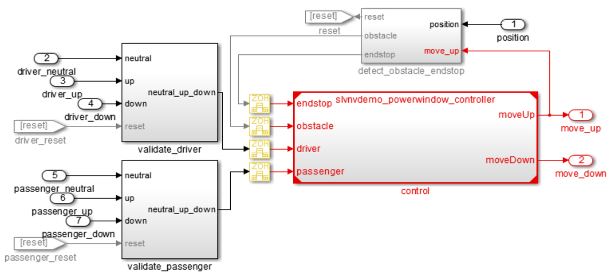


Figure 1: A slvnvdemo power window controller with validated passenger

Above figure shows the model which indicates about validated passenger in railway system.

ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 6, Issue 8, August 2017

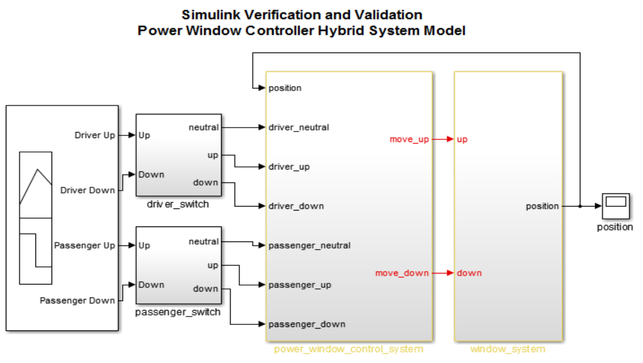


Figure 2: A slvnvdemo power window controller hybrid system model with passenger Up and down

Figure 2 indicate the power window controller for passenger Up and down position.

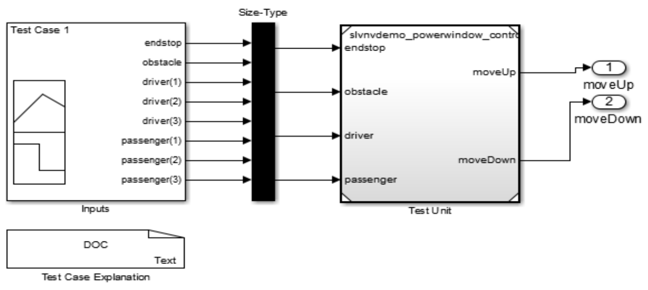


Figure 3: Test case explanation and unit testing of passenger for test case 1

In presented above figure shows the Test case for 1. It also indicates the Move up and down

position of passenger.

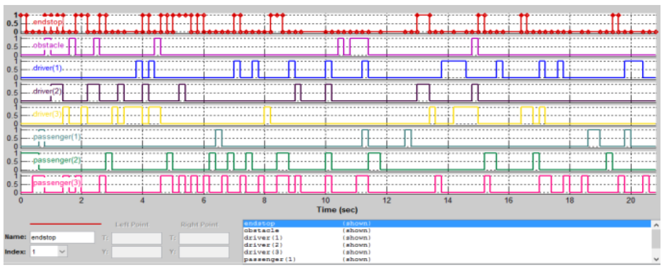


Figure 4: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

Now the very precious part of the Railway system, it indicated the signal of end stop, obstacle and passenger position

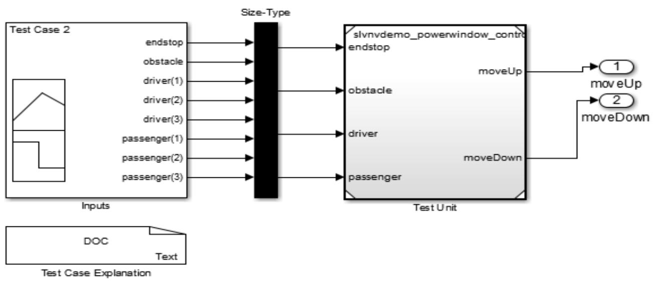
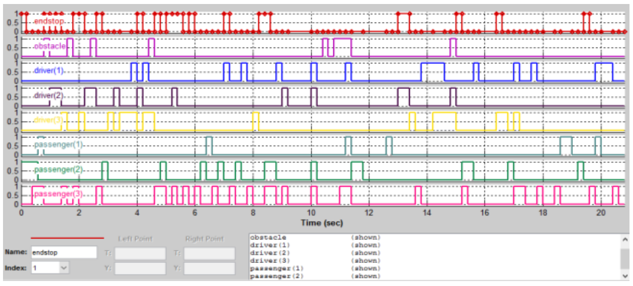


Figure 5: Test case explanation and unit testing of passenger for test case 2

As also discussed in figure 3, figure indicate the test case for 2 in software.



```
obstacle      (shown)
driver(1)     (shown)
driver(2)     (shown)
driver(3)     (shown)
passenger(1)  (shown)
passenger(2)  (shown)
```

Figure 6: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

The model starts by logging input signals to the component implementing the controller in its parent model and creating harness model for the controller from that logged data. A new test case in the harness model it captures all test cases and simulates the controller model for model coverage. Finally, it executes the controller with those test cases in simulation mode and Software-In-the-Loop (SIL) mode.

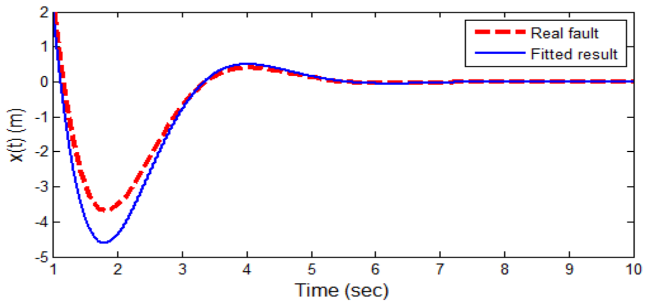


Figure 7: Real and fitted data

Above figure shows the fitted data and real fault. During the fitted data it also get some fault as shown in 7. Measures the percentage of the total variation about the mean accounted for by the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model does not fit the data well. The larger, the better the model explains the variation in the data

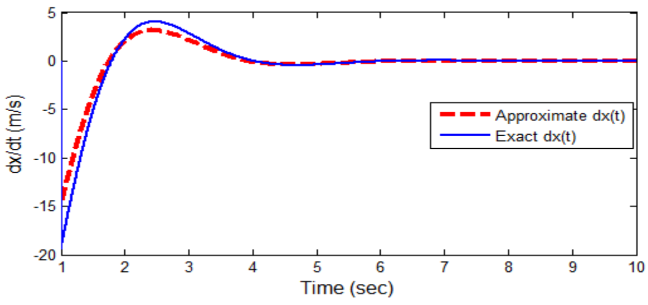


Figure 8: Approximate dx (t) and Exact dx (t)

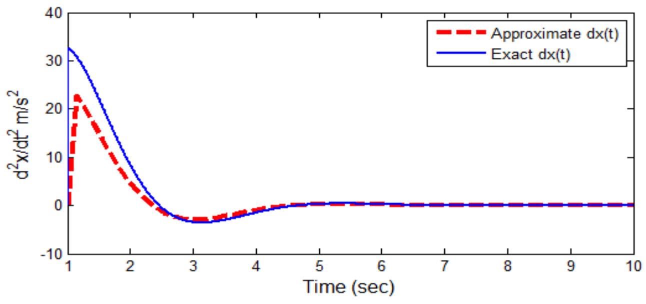


Figure 9: Approximate dx (t) and exact dx (t) on time

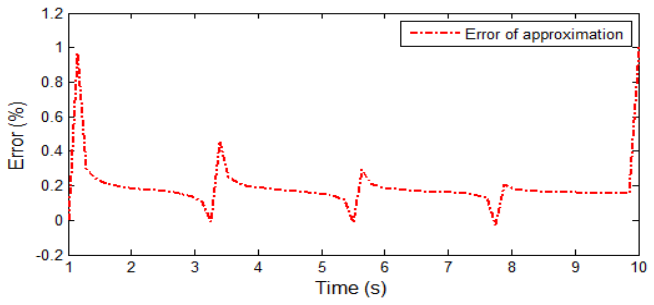


Figure 10: Error of approximation of software prediction

The error approximation incorporating the effect of time and covariates simultaneously and Sometimes, the effect of covariates are insignificant and the reduced form of the model may prove to be a better fit for the data and this can easily be obtained by setting f = 0, which gives us the better model. Similarly, when both covariate effect and time trend are insignificant, the model reduces to a NHPP model, with a constant recurrence rate, a. The number of intervals is always less or equal to number of failures that we observed because there can be more than one failure in any time interval.

7. Conclusion

In this paper, a new reliability model addressing the testing coverage is presented. Testing coverage is a very important measure for both software developers and users. The sets of software testing data have been used to examine the goodness-of-fit of all models. The criteria have been used to compare the models and the results show that the new model fits significantly better using Naïve bayes prediction method. Optimal software release policies are also achieved and it can be used to determine when to stop testing the software so as to minimize the expected total cost and satisfy the software reliability requirements. Consider a Poisson process, (N(t)),$t\geq 0$ with non-homogeneous intensity $\lambda(t)$. Here, we consider a deterministic function, not a stochastic intensity. Our model is helpful in Railway fault prediction through software management.

References

[1.] W. Burkhart, Z. Fatiha "Testing Software and Systems" // 23rd IfipWg 6.1 International Conference (2011), 236.

[2.] K. Goseva-Popstojanova, A.P. Mathur, K.S. Trivedi "Comparison of architecture–based software reliability models" // 12th International Symposium on Software Reliability Engineering (2001), 22-31.

[3] H. Okamura, Y. Etani, T. Dohi. A multi-factors software reliabilitymodel based on logistic regressionProc. of the 21stInternational Symposium on Software Reliability Engineering, 2010: 31−40.

[4] P. K. Kapur, H. Phan, S. Anand, et al. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans. OnReliability, 2011, 60(1): 331−340.

[5] C. Y. Huang, S. Y. Kuo. Analysis of incorporating logistic testing-effort function into software reliability modelingIEEETrans on Reliability, 2002, 51(3): 261−270.

[6] J. D. Musa, A. Iannino, K. Okumoto. Software reliability, measurement, prediction and application New York:McGraw-Hill, 1987.

[7] P. K. Kapur, D. N. Goswami, A. Gupta. A software reliability growth model with testing effort dependent learning function for distributed systemsInternational Journal of Reliability,Quality and Safety Engineering, 2004, 11(4): 365−377.

[8] S. Y. Kuo, C. Y. Huang, M. R. Lyu. Framework for modeling software reliability, using various testing-efforts and fault-detection ratesIEEE Trans. on Reliability, 2001, 50(3): 310−320.

[9] N. Ahmad, M. U. Bokhari, S. M. K. Quadri, et al. The exponentiatedWeibull software reliability growth model with various testing-efforts and optimal release policyInternationalJournal of Quality and Reliability Management, 2008, 25(2): 211−235.

[10] M. U. Bokhari, N. Ahmad. Analysis of a software reliability growth models: the case of log-logistic test-effort function. Proc. of the 17th IASTED International Conference on Modelingand Simulation, 2006: 540−545.

[11] C. T. Lin, C. Y. Huang. Software reliability modeling with Weibull-type testing-effort and multiple change-pointsProc.of the IEEE Region 10 Conference, 2005 (in CD format)

[12] C. Y. Huang. Performance analysis of software reliability growth models with testing-effort and change-point. Journalof Systems and Software, 2005, 76(2): 181−194.

[13] J. Zhao, H. W. Liu, G. Cui, et al. Software reliability growth model with change-point and environmental function. Journalof Systems and Software, 2006, 79(11): 1578−1587.

[14] C. T. Lin, C. Y. Huang. Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. Journal of Systems and Software, 2008, 81(6): 1025−1038.

[15] L. Fiondella, S. S. Gokhale. Software reliability model with bathtub-shaped fault detection rateProc of the Reliabilityand Maintainability Symposium, 2011, F: 24−27.

[16] S. Yamada, M. Ohba, S. Osaki. S-shaped reliability growth modeling for software error detectionIEEE Trans. on Reliability, 1983, R-32(5): 475−484.

[17] RazeefMohd,MohsinNazir,September 2012,"Software Reliability Growth Models: Overview and Applications",Journal of Emerging Trends in Computing and Information Sciences,Vol.3, No.9, pp-1309-1320.

[18] B.Anniprincy&Dr.S.Sridhar ,2014 "Two Dimasional Software Reliability Growth Models Using Cobb-Douglas Production Function and Yamada S-Shaped Model", Journal Of Software Engineering and Simulation ,Vol.2,Issue.2,pp-01-11.

[19] Hoang Pham, Xuemei Zhang, 2003"NHPP Software Reliability and cost models with testing coverage", ELSEVIER ,pp.443-454.