IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Analysis of Various Software Reliability Growth Models and Neural Networks Techniques used for Enhancement in Software Reliability

Ms. Roopa, Assistant Professor, Department of Computer Science & Applications, Ganga Institute of Technology & Management, V.P.O. Kablana, Jhajjar.

Abstract:
Software reliability is a key part in software quality. It is the ability of the software to perform its specified function under some specific condition. Reliability can be associated with both hardware and software. The hardware reliability can easily be evaluated since hardware get wear out but in case of software it be very difficult. So neural networks have been used for the last few decades in a broad variety of applications. In this paper we have studied the software reliability of neural network based and made a review on comparative analysis of existing software reliability growth models. The description and demonstration of the several types of neural networks are given, applications of neural networks like ANNs in medicine are described, and the detailed historical backgrounds are provided. The relationship between the artificial and the actual thing is also thoroughly investigated and explained.

Keywords:

Software Reliability, Reliability Model, Neural Network. Mean Value Function, Failure Intensity Function.

Introduction:

Reliability in the general engineering sense is the probability. It gives component or system in a define environment will operate correctly for a specified period of time. An important issue in developing such software systems is to produce high quality software system that satisfies user requirements. Software reliability models specify some reasonable form for this distribution, and are fitted to data from a software project. Once a model demonstrates a good fit to the available data, it can be used to determine the current reliability of the software, and predict the reliability of the software at future times.

Software reliability is often defined as ―the probability of failure-free operation in a defined environment for a specified period of time. A failure is the departure of software behavior from user requirements. This dynamic phenomenon has to be distinguished from the static fault (or bug) in the software code, which causes the failure occurrence as soon as it is activated during program execution.

Since software does not deprecate like hardware, the reliability of software stays constant over time if no changes are made to the code or to the environmental conditions including the user behavior. However, if each time after a failure has been experienced the underlying fault is detected and perfectly fixed, and then the reliability of software will increase with time. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability. The main problem is that the software systems are complex so that software engineers are not currently able to test software well enough to insure its correct operation. Due to the assumptions made by various software reliability models, or due to there is dependence among successive software runs . These problems are:

i) By finding mechanisms or relationships to more accurately determine the quality of software systems, without visiting a large fraction of their possible states.

ii) Taking in consideration the failure correlation and;

iii) Considering there is no single model sufficiently trustworthy in most or all applications.

There are many ways of using parametric models, nonlinear time series analysis and data mining to model software reliability and quality have been investigated. Fuzzy logic, neural networks, genetic algorithm, genetic programming and evolutionary computation are the most important key methodologies. Failure and fault are two different factors which are generally inbuilt in our software during the development phase. Fault can be said as an error or bug which is introduced during the development phase. These investigations point the way towards using computational intelligence technologies to support human developers in creating software systems by exploiting the different forms of uncertainty present in a software system results from infrequent and unpredictable occurrence of human errors and incomplete or imprecise data, in order to model complex systems and support decision making in uncertain environments. Software Reliability is an important attribute of software quality, functionality, usability, performance, serviceability, capability, install ability, maintainability, and documentation. It is hard to achieve, because the complexity of software tends to be high. There are three major classes of software reliability:

1) **Black box reliability analysis**: Estimation of the software reliability based on failure observations from testing or operation. These approaches are called black box approaches because internal details of the software are not considered.

2) **Software metric based reliability analysis:** Reliability evaluation based on the static analysis of the software (e.g., lines of code, number of statements, complexity) or its development process and conditions (e.g., developer experience, applied testing methods).

3) **Architecture-based reliability analysis:** Evaluation of the software system reliability from software component reliabilities and the system architecture (the way the system is composed out of the components). These approaches are sometimes called component-based reliability estimation (CBRE), or grey or white box approaches.

Software reliability is about to define the stability or the life of software system with different properties. These properties include the trustfulness of software system, software cost, execution time, software stability etc. The aspects related to these software system includes the probability of software faults, frequency of fault occurrence, criticality of fault, associated module with respective fault etc. In a software development process, the pre estimation of software reliability is required to deliver the software product. According to the required level of software quality estimation of software cost, development time is also estimated. There are number of quality measure that approves the software reliability **[1]**. Each stage of software life cycle itself takes some time quantum to deal with software reliability. Higher the software quality, lesser the software maintainability. Software reliability growth models, refers to those models that try to predict software reliability from test data **[2]**. These models try to show a relationship between fault detection data (i.e. test data) and known mathematical functions such as logarithmic or exponential functions. The goodness of fit of these models depends on the degree of correlation between the test data and the mathematical function **[3]**.

The software reliability is defined as the probability that the software will operate without a failure under a given environmental condition during a specified period of time **[4]**. The software reliability assessment is one of the most important processes during the software development. Since 1970, many software reliability growth models (SRGMs) have been proposed. In general, there are two major types of software reliability models: the deterministic and the probabilistic. The deterministic one is employed to study the number of distinct operators and operands in the program. The probabilistic one represents the failure occurrences and the fault removals as probabilistic events **[5]**.

The probabilistic models can be further classified into different classes, such as error seeding, failure rate, and non-homogeneous Poisson process (NHPP). Among these classes, the NHPP models are the most popular ones. The reason is the NHPP model has ability to describe the software failure phenomenon. The first NHPP model, which strongly influences the development of many other models presented a NHPP model with S-shaped mean value function. They **[6, 7]** also made further progress in various S-Shaped NHPP models. Although these NHPP models are widely used, they impose certain restrictions or a priori assumptions about the nature of software faults and the stochastic behavior of software failure process.

Role of Neural Network in Software Reliability:

The reliability model till now been designed are based on the study of failure associated with the code and the environment where it is been implemented. All the software reliability models are designed on the basis of execution time and calendar time. Execution time of any program is the time that is actually required or spent by the processor in executing the instruction of that program .Calendar time is referred as the elapsed time from start to end of program execution on a running computer.

Neural network is a collection of fast processing and computing nodes called artificial neurons. These neurons are designed on the basis of study of the behavior of biological neuron. These neurons are connected in a specific manner that is layer like structure known as neural network architecture. Neural network is an output based computing technique. It is a technology generally been used for optimizing problem. Neural networks also consist of multiple layers of computational units, usually interconnected in a feed-forward way. Neural network has been applied to estimate parameters of the formal model and to learn the process itself in order to predict the future outcomes. It has been shown that feed forward network can be applied for prediction. Back-error propagation is one of the most widely used neural network paradigms and has been applied successfully in application studies in a broad range of areas. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function. ANN (Artificial Neural Network) software reliability models have recently aroused more research interest. Traditionally, both kinds of models only consider single fault detection process (FDP) and data for analysis are only from FDP. However, while data from both FDP&FCP (fault correction process) are available, NHPP and ANN models can be extended into paired NHPP models and combined ANN models, providing more accurate predictions.

The basic characteristics of a neural network **[8]** are

- It consists of many simple processing units, called neurons that perform a local computation on their input to produce an output.
- Many weighted neuron interconnections encode the knowledge of the network.
- The network has a learning algorithm that lets it automatically develop internal representations

One of the most widely used processing unit models is based on the logistic function. The resulting transfer function is given by

$$output = \frac{1}{1 + e^{-sum}}$$

Where sum is the aggregate of weighted inputs. Figure 1 shows the actual I/O response of this unit model. The unit is nonlinear and continuous. There exists a variety of neural network models and learning procedures. Two well-known classes of neural networks that can be used for prediction applications are: feed-forward networks and recurrent networks. They use feed-forward networks and learning procedure is back propagation algorithm which comes under the category of supervised learning.

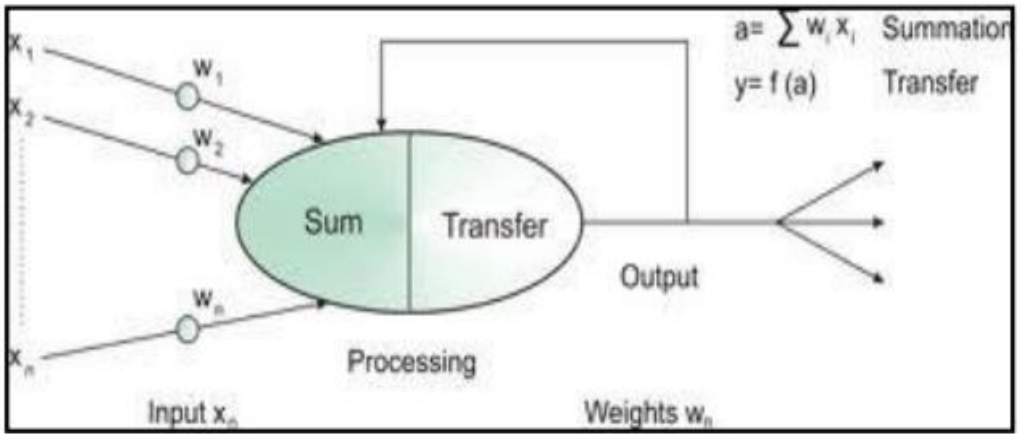

Fig. 1 A typical transfer unit

Architecture Of Neural Network:

The artificial neural network is designed on the basis of the study of biological neural network. A human brain is responsible for all the action and reaction taken by human body. These actions are controlled by brain and are carried to different parts of the body by a fast processing node known as neurons. Similarly in case of artificial neurons also we have a fast processing node responsible for performing the computation and are known as Neurons [19]. A neural network is a collection one to multiple neurons which are arranged in a specific manner to perform the computation. The basic structure of a neuron is represented in the given below figure:

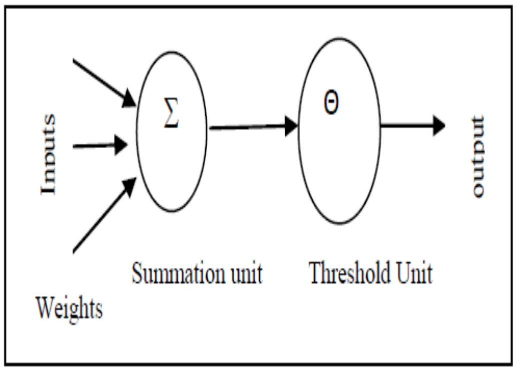

Figure2: Structure of a Neuron

The model is designed for a set of input values and corresponding desired output. Suppose consider that „I‟ is the set of input values such that I={i1,i2…….in}and we associate a variable parameter with it which controls the behaviour3 of neurons. Let us consider weights associated with the input as the variable parameter such that W={w1,w2………wn}.All the input along with the weight is passed through the summation unit where compute the net input which given by

$$\text{Net}_{\text{input}} = \sum_{i=1}^{n} IiWi \text{ where n = number of input to the network} \;\ldots\ldots..(1.11)$$

$$= i1*w1+i2*w2+i3*w3+\ldots\ldots.+i_i wi$$

These neurons are arranged in different layers to represent a specific structure known as neural network architecture. For designing a reliability model we will require a multilayer of neurons. The neurons are arranged in three different layers known as input layer, hidden layer and output layer. The multi layer network is shown in below Figure2:

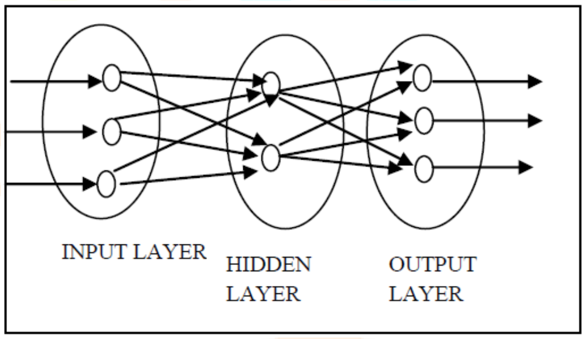

Figure3: Multi-Layer Neural Networks

Back Propagation Algorithm

It is the training or learning algorithm. Example plays a huge role here. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights so that it can produce required output for a particular input after completing the training.

Back Propagation networks are useful for simple Pattern Recognition and Mapping Tasks.

Bayesian Networks (BN)

These are also known as Belief Networks or Bayes Nets .They are the graphical structures used to represent the probabilistic relationship among a set of random variables. BNs reason about the uncertain domain.

Each node represents a random variable with specific propositions in these networks. For example, in a medical diagnosis domain, the node Cancer represents the proposition that a patient has cancer. The edges connecting the nodes represent probabilistic dependencies among those random variables. If out of two nodes, one is affecting the other then they must be directly connected in the directions of the effect. The strength of the relationship between variables is measured by the probability associated with each node.

The only constraint on the arcs in a BN is that you cannot return to a node simply by following directed arcs sufficing for the BNs to be called Directed Acyclic Graphs (DAGs).

BNs are capable of handling multivalued variables simultaneously. The BN variables are composed of two dimensions −

Applications of Neural Networks

They can perform tasks that are easy for a human but difficult for a machine −

- Aerospace − Autopilot aircrafts, aircraft fault detection.
- Automotive − Automobile guidance systems.
- Military − Face recognition Weapon orientation and steering, target tracking, object discrimination, signal/image identification.
- Electronics − Code sequence prediction, IC chip layout, chip failure analysis, machine vision, voice synthesis.
- Financial − Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, portfolio trading program, corporate financial analysis, currency value prediction, document reading, credit application evaluating.
- Industrial − Manufacturing process control, product design and analysis, quality inspection systems, welding quality analysis, paper quality prediction, chemical product design analysis, dynamic modelling of chemical process systems, machine maintenance analysis, project bidding, planning, and management.
- Medical − Cancer cell analysis, EEG and ECG analysis, prosthetic design, transplant time optimizer.
- Speech − Speech recognition, speech classification, text to speech conversion.
- Telecommunications − Image and data compression, automated information services, real-time spoken language translation.
- Transportation − Truck Brake system diagnosis, vehicle scheduling, routing systems.
- Software − Pattern Recognition in facial recognition, optical character recognition, etc.
- Time Series Prediction − Predictions on stocks and natural calamities.
- Signal Processing − Process an audio signal and filter it appropriately in the hearing aids.

- Control − Make steering decisions of physical vehicles.

- Anomaly Detection – Since ANNs are expert at recognizing patterns, they can also be trained to obtain an output when something undesirable occurs that mismatches the pattern.

Advantages:

- A neural network can perform tasks which a linear program cannot perform.
- Due to their parallel nature, even when an element of the neural network fails, it can continue without any problem.
- A neural network need not be reprogrammed as it learns itself.
- It can be implemented in an easily without any problem.
- As the adaptive, intelligent systems, neural networks are robust and expert at solving complex problems. They are efficient in their programming and the scientists agree that the advantages of using ANNs outweigh the risks.
- It can be implemented in any type of application.

Disadvantages:

- The neural networks need training to operate.
- High processing time is required for large neural networks.
- The architecture of a neural network is different from the architecture of microprocessors. So, they have to be emulated.

SOFTWARE RELIABILITY GROWTH MODELS (SRGMS) AND CRITERIA

A software reliability growth model (abbreviated as SRGM) is known as one of the fundamental technologies for quantitative software reliability assessment, and playing an important role in software project management for producing a highly-reliable software system**[9]**. SRGM is mathematical model, shows how software reliability improves as faults are detected and repaired. SRGM can be used to predict when a particular level of reliability is likely to be attained. Thus, SRGM is used to determine when to stop testing to attain a given reliability level **[10]**. There are many software reliability growth models but the commonly used model of software reliability models are JM, GO model, MO model, Sch model, S-Shape model. To evaluate the prediction powers of different models, it is necessary to use a meaningful measures. They use two criteria: Root Mean Square Error (RMSE) and Average Error(AE). These criteria are used to measure the difference between the actual and predicted values, the formulas is as Eq. (1) (2).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(c(k) - \hat{c}(k))^2} \qquad (1)$$

$$AE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{c(k)-\hat{c}(k)}{c(k)}\right| \times 100 \qquad (2)$$

Where n is the number of groups of failure data, $c(k)$ is the number of the actual failures in each group of failure data, $\hat{c}(k)$ is the number of the predicted failures. The smaller the RMSE and AE, the stronger that the model prediction ability **[2]**.

A. *Various types of Models*

1) *Jelinski-Moranda Model:* The Jelinski-Moranda model was first introduced as a software reliability growth model in Jelinski and Moranda (1972) **[11]**. This is a continuous time-independently distributed inter failure times and independent and identical error behavior model **[12]**. The software failure rate of hazard function at any time is proportional to the current fault content of the program. The distribution of the order statistics is the Exponential distribution **[11]**.

The main assumptions for the Jelinski-Moranda model are the following:

(1) At the beginning of testing, there are n_0 faults in the software code with n_0 being an unknown but fixed number.
(2) All faults are of the same type.
(3) Immediate and perfect repair of faults.
(4) Faults are detected independently of each other.
(5) The times between failures are exponentially distributed with parameter proportional to the number of remaining faults.
(6) Each fault is equally dangerous with respect to the probability of its instantaneously causing a failure. Furthermore, the hazard rate of each fault does not change over time, but remains constant at φ.
(7) The failures are not correlated, i.e. given n_0 and φ the times between failures $(\Delta t_1, \Delta t_2, \ldots, \Delta t n_0)$
(8) Whenever a failure has occurred, the fault that caused it is removed instantaneously and without introducing any new fault into the software.

$$z(\Delta t|t_{i-1}) = \phi[n_0 - M(t_{i-1})] = \phi[n_0 - (i-1)] \qquad (1)$$

The failure intensity function is the product of the inherent number of faults and the probability density of the time until activation of a single fault, $n_a(t)$, i.e.:

$$\frac{dm(t)}{d(t)} = n_0[1 - \exp(-\phi t)] \qquad (2)$$

Therefore, the mean value function is

$$m(t) = n_0[1 - \exp(-\phi t)] \qquad (3)$$

It can easily be seen from equations (2) and (3) that the failure intensity can also be expressed as

$$\frac{dm(t)}{d(t)} = \phi[n_0 - m(t)] \qquad (4)$$

According to equation (4), the failure intensity of the software at time t is proportional to the expected number of faults remaining in the software; again, the hazard rate of n individual faults is the constant of proportionality.

Moreover, many software reliability growth models can be expressed in a form corresponding to equation (4).

One of the most widely discussed assumptions of the Jelinski-Moranda model is (2) since it implies that each repaired fault reduces the hazard rate of the new time between failure by a constant $\lambda > 0$. This idea is depicted in Figure 2.

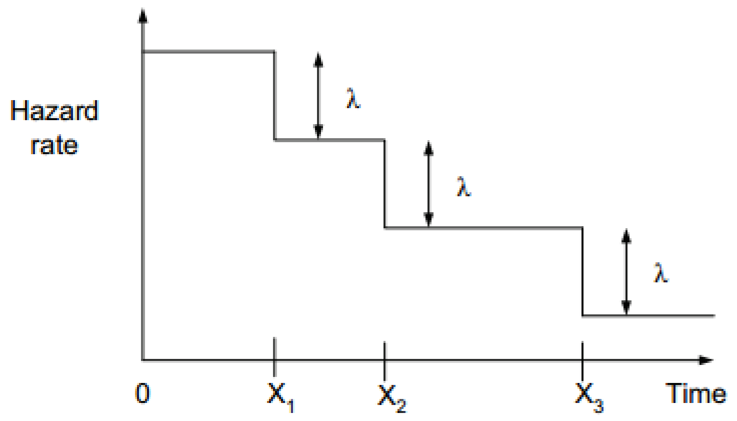

Fig. 4 Jelinski- Moranda model hazard rate. It remains constant between failure observations and decreases by a factor λ after a fault is repaired [11]

2) 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. Assumptions (2), (3) and (4) for the Jelinski-Moranda model are also valid for the Goel-Okumoto 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 **[13]**

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

and the intensity function of this model is given as

$$\lambda(t) = ab * \exp[-bt], \qquad a > 0, b > 0$$

where a is the expected total number of faults to be eventually detected and b represents the fault detection rate.

In fact, it follows that **[11]**

$$\lim_{t\to\infty} m(t) = a$$

A typical plot of $m(t)$ for the Goel-Okumoto model can be observed in Figure 3 where $m(t)$ is plotted when $a = 11$ and $b = 0.14$. Note that a determine the scale and b the shape of the mean-value function.

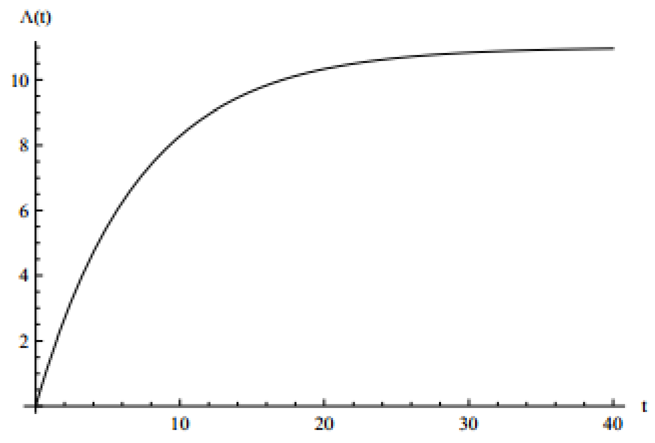

Fig 5: Goel- Okumoto model mean-value function when a = 11 and b = 0.14 [11]

3) *Generalized Goel NHPP Model*

In order to describe the situation that software failure intensity increases slightly at the beginning and then begins to decrease, Goel proposed a simple generalization of the Goel-Okumoto model with an additional parameter c **[13]** . The mean value function and intensity function are

$$m(t) = a(1 - \exp[-bt^c]), \qquad a > 0, b > 0, c > 0$$

$$\lambda(t) = abct^{c-1} \exp[-bt^c], \qquad a > 0, b > 0, c > 0$$

where a is the expected total number of faults to be eventually detected and b and c are parameters that reflect the quality of testing.

4) *Inflected S-Shaped Model*

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 **[13]**

$$m(t) = a * \frac{1 - \exp[-bt]}{1 + \psi(r) * \exp[-bt]}, \qquad \psi(r) = \frac{1 - r}{r}, \qquad a > 0, b > 0, r > 0$$

The parameter r 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. On taking $\psi(r) = \beta$ then the inflection S-shaped model mean value function and intensity function are given as

$$m(t) = a * \frac{1 - \exp[-bt]}{1 + \beta * \exp[-bt]}, \qquad a > 0, b > 0, \beta > 0$$

$$\lambda(t) = \frac{ab\exp[-bt](1+\beta t)}{(1+\beta * \exp[-bt])^2}, \qquad a>0, b>0, \beta>0$$

5) *Logistic Growth Curve Model*

In general, software reliability tends to improve and it can be treated as a growth process during the testing phase. That is, the reliability growth occurs due to fixing faults. Therefore, under some conditions, the models developed to predict economic population growth could also be applied to predict software reliability growth. These models simply fit the cumulative number of detected faults at a given time with a function of known form. Logistic growth curve model is one of them and it has an S-shaped curve. Its mean value function and intensity function are **[13]**

$$m(t) = \frac{a}{1+\mathrm{k} * \exp[-bt]}, \qquad a>0, b>0, \mathrm{k}>0$$

$$\lambda(t) = \frac{ab\exp[-bt]}{(1+\mathrm{k} * \exp[-bt])^2}, \qquad a>0, b>0, \mathrm{k}>0$$

where a is the expected total number of faults to be eventually detected and k and b are parameters which can be estimated by fitting the failure data.

6) *Musa-Okumoto Model*

Model Name	Mean Value Function	Intensity Function
1. Jelinski-Moranda model	$m(t) = n_0[1-\exp(-\phi t)]$	$\lambda i = (N-k)\mu$
2. Goel-Okumoto Model	$m(t) = a(1-exp[-bt]), a>0, b>0$	$\lambda(t) = ab * exp[-bt], a>0, b>0$
3. Generalized Goel NHPP Model	$m(t) = a(1-exp[-bt^c]),$ $a>0, b>0, c>0$	$\lambda(t) = abct^{c-1} exp[-bt^c],$ $a>0, b>0, c>0$
4. Inflected S-Shaped Model	$m(t) = a * \frac{1-exp[-bt]}{1+\psi(r) * exp[-bt]},$ $\psi(r) = \frac{1-r}{r}, \quad a>0, b>0, r>0$	$\lambda(t) = \frac{abexp[-bt](1+\beta t)}{(1+\beta * exp[-bt])^2},$ $a>0, b>0, \beta>0$
5. Logistic Growth Curve Model	$m(t) = \frac{a}{1+k * exp[-bt]},$ $a>0, b>0, k>0$	$\lambda(t) = \frac{abexp[-bt]}{(1+k * exp[-bt])^2},$ $a>0, b>0, k>0$
6.Musa-Okumoto Model	$m(t) = a * ln(1+bt), \quad a>0, b>0$	$\lambda(t) = \frac{ab}{(1+bt)}, \quad a>0, b>0$

Musa-Okumoto have been observed that the reduction in failure rate resulting from repair action following early failures are often greater because they tend to the most frequently occurring once, and this property has been incorporated in the model **[13]**. The mean value function and intensity function of the model given as

$$m(t) = a * \ln(1 + bt), \qquad a > 0, b > 0$$

$$\lambda(t) = \frac{ab}{(1 + bt)}, \qquad a > 0, b > 0$$

where a is the expected total number of faults to be eventually detected and b is the fault detection rate.

PROPOSED MODEL

On the basis of the model discussed the logistic growth curve model seems perform better than the other models. The model can be designed on the basis of the tangential function. The tangential model must be drawn in the positive axis. It varies from 0 to infinity similar to the software reliability. The software reliability is inversely proportional to the fault detection as the no of fault detection decrease the reliability increases. The zero fault detection means the infinite reliability and the zero software reliability means the infinite faults. The proposed model suits the behavior of the software reliability so fits to the software reliability.

In the beginning of testing, there is exponential number of faults in the software code. The number of faults is unknown but they are fixed in number. All faults are of same type. Each fault can be detected independent of each other. The remaining number of fault and the remaining time is useful to determine the other parameters. The probability of occurring of each fault is same. Each fault occurred can be removed instantaneously. The mean value function can be given as

$$m(t) = \left(fv = \left(\frac{[1-\exp(-\phi t)]}{[1+\exp(-\phi t)]}\right)\right) > 0 ? fv : 0 \qquad (3)$$

The failure intensity can be expressed as

$$\lambda(t) = \frac{dm(t)}{d(t)} \qquad (4)$$

According to the failure intensity of the software at time t is proportional to the expected number of faults remaining in the software.

Conclusions:

The main purpose of this paper was to understand the concept of software reliability using the artificial neural network. This paper introduces the concept of neural model and its architecture. In future we are going to design a neural model for calculating the reliability. A neural network is considered as an optimizing technique which is used to scale the output. When a software system is designed, the major concern is the software quality. The quality of software depends on different factors such as software reliability, efficiency, cost etc. This paper study various existing software reliability model with their failure intensity function and the mean value function. This paper also proposes a new model for software reliability having different failure intensity function and mean value function. In future the proposed can be implemented and results of the model can be compared with the existing model results.

References:

[1] Garima Chawla et. al. , *A Fault Analysis based Model for Software Reliability Estimation*, International Journal of Recent Technology and Engineering (IJRTE),ISSN: 2277 -3878, Volume-2, Issue-3, July 2013.

[2] J. Laprie and K. Kanoun "Software Reliability and System Reliability", In M. R. Lyu, editor, Handbook of Software Reliability Engineering, pages 27–69. McGraw Hill, 1996.

[3] K. Goseva-Popstojanova, and K. Trivedi "Failure Correlation in Software Reliability Models", IEEE Transactions on Reliability, Vol. 49, No. 1, March 2000.

[4] S., Dick and A. Kandel "Computational Intelligence in Software Quality Assurance", World Scientific Publishing Co. 2005.

[5] Rita G. Al gargoor et. al. , *Software Reliability Prediction Using Artificial Techniques*, IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, No 2, July 2013,ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784,www.IJCSI.org.

[6] Al-Rahamneh Z., Reyalat M. Sheta A. F., Bani-Ahmad S., and Al-Oqeili S., *A New Software Reliability Growth Model: Genetic-Programming-Based Approach* , Journal of Software Engineering and Applications, 2011, pp. 476-481.

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

[8] Su, Yu Shen, et al. *An Artificial Neural-Network-Based Approach To Software Reliability Assessment*, TENCON 2005, IEEE Region 10. IEEE, 2005

[9] S. Yamada, and S. Osaki, *Reliability Growth Models for Hardware and Software Systems Based on Non-homogeneous Poisson Processes: a Survey*, Microelectronics and Reliability, 23, 1983, pp. 91-112.

[10] S. Yamada, and S. Osaki, *S-shaped Software Reliability Growth Model with Four Types of Software Error Data*, Int. J. Systems Science, 14, 1983, pp. 683-692

[11] Inoue S., and Yamada S ,*Two-Dimensional Software Reliability Measurement Technologies*, IEEE , 2009, pp. 223 – 227

[12] Quadri S. M., Ahmad N. and Farooq S. U. ,*Software Reliability Growth Modeling With Generalized Exponential Testing –Effort And Optimal Software Release Policy* , Global Journal of Computer Science and Technology , 2011, pp.27 – 42

[13] Prof. C. J. van Duijn et. al. ,*Statistical Procedures for Certification of Software Systems*, © Corro Ramos, Isaac 2009.

[14] Latha Shanmugam et. al., *A Comparison Of Parameter Best Estimation Method For Software Reliability Models*, International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.5, September 2012.

[15] Mohd. Anjum et. al. , *Analysis and Ranking of Software Reliability Models Based on Weighted Criteria Value*, I.J. Information Technology and Computer Science, 2013, 02, 1-14.

[16] Mamta Arora et. al. , *Software Reliability Prediction Using Neural Network*, International Journal of Software and Web Sciences 5(2), June-August, 2013, pp.88-92.

[17] Musa JD " Validity of the Execution time theory of Software Reliability "IEEE trans on Reliability R-283) pp 181-191 Aug1979.

[18] Eckhardth D.E et al "An experimental Evaluation of Software redundancy as a strategy for improving reliability" IEEE trans on software engineering.

[19] By KK Agarwal and Yogesh Singh "Software Engineering"

[20] M.R. Lyu, "Handbook of Software Reliability Engineering", *IEEE Computer Society Press*, McGraw Hill, 1996.

[21] K. Mehrotra, C.K. Mohan and S. Ranka, "Elements of Artificial Neural Networks", *Penram International Publishing*, 1997.

[22] Tian, L. & Noore, A., "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering & System Safety*, vol. 87, no. 1, pp. 45-51, 2005.

[23] Guo, P. & Lyu, M.R., "A pseudo inverse learning algorithm for feed forward neural networks with stacked generalization applications to software reliability growth data," *Neurocomputing*, vol. 56, pp. 101-121, 2004.

[24] Ho, S.L., Xie, M. & Goh, T.N., "A study of the connectionist models for software reliability prediction," *Computers & Mathematics with Applications*, vol. 46, no. 7, pp. 1037-1045,2003.

[25] Cai, K.Y., Cai, L., Wang, W.D., Yu, Z.Y. & Zha ng, D, "On the neural network approach in software reliability modeling", *Journal of Systems and Software*, vol. 58, no. 1, pp. 47-62,2001.