International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

A Brief Study and Analysis of Software Reliability

Bishwajeet Kumar[1], Dr. I. B. Lal[2]

[1]*Research Scholar, B.R.A.Bihar University, Muzaffarpur, Bihar, India*
[2]*Assistant Professor, Deptt. of Information Technology, L.N. Mishra College of Business Management, Muzaffarpur, India*

Abstract:- **Software reliability has been a crucial issue for last two decades as computers have been penetrating every walk of human life, from kitchen to war field, from machine to market, and much more. The software is changing the human life style enormously throughout the world. Hence, delivery of highly reliable fault free software has become a challenging issue for the software professionals. Software engineers have been working on software reliability for last four to five decades. New techniques to study, predict, and estimate the software reliability are evolving day-by-day, such as fuzzy logic, neural networks, genetic algorithms, etc. Various models have been proposed in this regard. The authors have attempted to study and present an analysis on the software reliability.**

Keywords:- **Software reliability, fault, failure, software reliability prediction, fuzzy logic, artificial neural network, genetic algorithm.**

I. INTRODUCTION

With the increased applicability and usability throughout the world the software and its development as per real world modeling have become very much complex. The major reasons for this complexity can be noted as follows:-

a) the size of software increased due to increased number of functionality that the software is expected to perform,
b) frequently changing requirements of the user/environment,
c) code restructuring,
d) entry of software in every walk of life, and
e) the expectation of users and dependencies of the user community for the fault- free operation with reduced response time.

These situations have lead the software reliability at stake with the increased requirements of users, sophistication in requirements and operations, reduced failures (possibly zero) and the fault free operation. All have made software life cycle phases very much complex and difficult. To develop highly reliable software that stands up to the expectations of the developer as well as users, various considerations have to be studied and evaluated during development of software.

In today's highly technical world there are numerous applications such as traffic control, satellite system and other life sensitive and critical systems that require hundred percent failure-free, fault-free software with the probability of occurring errors as "No errors in millions LOC (Line of code)". These are the systems that cannot tolerate a single error (fault). Software reliability is defined as the probability of failure-free operation of a software or computer program for a specified period in a specified environment. It can also be defined as up to what extent one depends on software for its functionality for a specified period. J.D. Musa *et. al.*(1987) defines Software reliability as "the probability of the execution of software without failure for some specified interval of natural unit of time. A Computer system or software is considered reliable if it functions as per its specifications and produces a correct set of output values for a given set of input values. For a computer system or a software system reliable operation is attained when all components of the system work according to specifications. Software reliability is also defined as *the probability of failure free software operation for a specified period of time in a specified environment.* It is one of the attributes of software quality, a multidimensional property including other customer satisfaction factors – functionality, usability, performance, serviceability, capability, installability, maintainability and documentation. Software reliability is generally considered to be the key factor of software quality because it quantifies the failures – which can make a powerful system inoperative. We notice three major components in the definition of software reliability: *failure*, *time*, and *operational environment.* Following are some terms used for defining software reliability (Lyu, Michael, R., 1995):-

1.1 Michael R. Lyu (1995) defines "A Software System as an interacting set of software subsystems that is embedded in a computing environment that provides inputs to the software system and accepts service (outputs) from the software". A software as a system is composed of number of programs or modules which interact with each other interdependently so as to perform its functionalities to produce the desired output.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

1.2 Service: A service is the delivery of outputs as per expectations of the user within the a given time period. A software consisting of set of programs or modules (or component), as a system, is also a time dependent sequence of outputs to an environment or a person – the user.

1.3 The failure of a system occurs if the system stops working or is unable to deliver its functionalities according to the specification and fails to deliver the services for which it was intended. In a software system the failure can also be observed if it is unable to deliver its functionalities or outputs according to the external environment or the user. In such cases the user stops using the software.

1.4 Outages: An outage can be defined as a loss or degradation of service to a user/customer for a period of time. This might occur due to human errors, change in technology, hardware malfunctioning or failure, software failures, or any other environmental factor that might affect the services or behaviour of the software system.

1.5 An error in the system occurs when a component of the system assumes a state that is not desirable or if the component deviates from the standards or set norm or the stated requirement or functionality. The components in question is undesirable and said to be erroneous and further use of the components will lead to failure. M.R.Lyu (95) defines "Error" as a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. Errors occur when some part of the computer software produces an undesired state.

1.6 A fault is detected either when an error is propagated from one component to another (Ripple effect) or the failure of the computer is observed. An error, when occurs in a system, propagates and enlarges in size and due to this, components of the system violates and fails to deliver standard and normal functionality. This observed state of error is called fault. A fault is in effect the identified or an assumed cause of error. It is also referred to as a bug.

1.7 Failure Functions: Lyu(95) describes failure function in following ways on the basis of time:

(i) The cumulative failure function – also referred to as mean value function and denotes the average cumulative failures associated with each point of time.

(ii) The failure intensity function represents the rate of change of the cumulative failure function.

(iii) The failure rate function – also called the rate of occurrence of failures, is defined as the probability that a failure per unit time occurs in the interval [t, t+, Δt], given that a failure has not occurred before t.

(iv) The mean time to failure (MTTF) function, also called mean time between failure (MTBF) is the average time between two failures. It represents the expected time that the next failure will be observed. *(Adopted from M.R.Lyu,1995).*

Problems of reliability have been a greater of concern for larger software systems. It has become bottleneck of system reliability, and the maturity of software always lags behind that of hardware. Accurately modeling software reliability and predicting its trend have become critical. Reliability engineering is routine practice in many engineering disciplines. Software Reliability Engineering (SRE) can be defined as *the qualitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability [IEEE95]*.

Thus, SRE must deal with the reliability of software based systems systematically with respect to user requirements. SRE, therefore, must take into account : User profile, operational environment, Requirements specification and requirement engineering, system architecture, software product design, software development process, testing, use and maintenance followed by reliability management that would encompass error / fault detection, reliability measurement including reliability estimation and prediction. The early prediction would certainly help reduce/eliminate problems of reliability.

Now-a-days, the development of software system has been related more to human factors besides hardware considerations. Statistical analysis and quality control would certainly improve the situation. Software Quality Assurance (SQA) and Statistical Process Control (SPC) have been considered by so many workers to enhance the reliability of the software. SPC concepts and methods are used to monitor the performance of a software process over time in order to verify that the process remains in the state of statistical control. It helps in finding assignable causes, long-term improvements in the software process. Software quality and reliability can be achieved by eliminating the causes or improving the software process or its operating procedures (Kimura, *et. al.*,2011). Number of models has been proposed by so many workers to enhance the reliability of the software. We have, in the foregoing discussions, attempted to study various aspects to increase software reliability.

II. Review Of Literature

Reliability of the software has been a major issue in the mid era of software engineering approaches. A large number of computer scientists have been working on the issue to improve the software development process and techniques to build software that is reliable and runs without failure.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Number of reliability growth models was proposed in this regard. In the early days, software reliability was considered as immeasurable attribute of software. This mis-conception was wiped out and various statistical methods were proposed. M.L. Shooman (1987) published an article *"Yes Software Reliability Can be Measured and Predicted"*. J. D. Musa (1975) postulated a theory of *"Software Reliability and its Applications"*. M. Ohba (1984) proposed various *"Software Reliability Analysis Models"*. S. Yamada, M.Ohba, and S.Osaki(1983) proposed *"S- Shaped Reliability Growth Modeling for Error Detection"* . J.D. Musa and K. Okumoto (1984) submitted *"A Logarithmic Poisson Execution Time Model for Software Reliability Measurement"*. Okamura and Dohi in 2006, introduced a method by using *"Expectation-Maximization (EM) principle and applied the same in Hybrid, Discrete time and Markov Modulated Software Reliability Models"*. Minohara and Tohma (in ISSRE 1995) introduced a Genetic Algorithm for Hyper-Geometric Distribution Software Reliability Growth Models and it was observed that length of coding string has been reduced by proportion coefficient; hence the parameter's searching range was decreased. Zhang in 2008 applied Particle Swarm Optimization (PSO) algorithm but it was observed that the searching range is too large and the convergence speed is slow and accuracy is not high. Ritika Wason (2012) proposed new paradigm for estimation by novel Finite Automata based software reliability model that implicitly scores over the traditional models on many factors, most importantly due to the fact that is based on the realistic assumption that a software system in execution is a Finite State Machine.

In 1972, Jelinski and Moranda worked on *reliability growth modeling* and developed a mathematical model, popularly known as J-M model, which is a continuous time-independently distributed inter failure times. The model which also assumes independent and identical error behavior forms the basis for many other SRGMs. A large number of researchers have used and considered this model. 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. This earliest SRGM supposes that failures occur according to the Poisson process with hazard rate decreasing as more faults are detected and successfully removed. In 1978 Schick and Wolverton in his studies modified this model which was based on suggesting increasing failure rate between successive failures instead of the previously suggested constant failure rate by Jelinski and Moranda.

Myron Hecht in 2006 described that for terrestrial and space elements software becomes a more important cause of operational failures. Claes Wholin in 2007 introduced three ways to estimate parameters of the model. He described that parameter can be evaluated by comparing historical data to previous data. Parameters can be estimated using information from the current project. In 2008, Leslie Cheung, Roshanak Roshandel, Nenad Medvidovic proposed a framework for predicting reliability of software components at architectural design.

III. Software Reliability Model

A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment. The figure (*Adopted from M.R.Lyu,95)* given below shows the basic ideas of software reliability modeling.

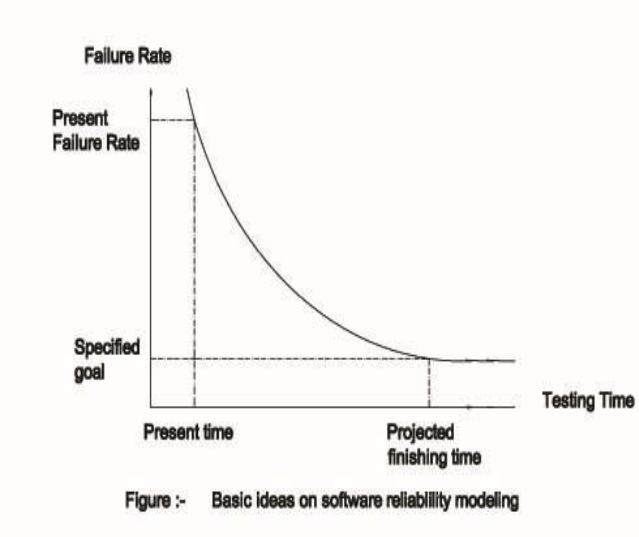

Figure :- Basic ideas on software reliability modeling

(Figure adopted from M.R.Lyu,95)

In the figure, the failure rate of a software system which is high in the early stage is generally decreasing due to fault/failure detection and removal. At any particular time it is possible to observe a history of the failure rate of the software. Software reliability modeling forecasts the curve of the failure rate by statistical evidence. The purpose of this measure is to predict the extra time needed to test the software to achieve a specified objective, and to predict the expected reliability after the the testing is finished.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Software reliability is similar to hardware reliability as both are stochastic processes and can be described by probability distributions. However, software reliability is different from hardware reliability in the sense that software does not wear out, i.e., its reliability does not decrease with time. However, software reliability decreases due to abrupt changes during operation or incorrect modifications to the software. During testing and operation faults are detected. Hence software reliability enjoys growth during this period because at the stage faults could be removed avoiding the failures.

In contrast to hardware faults which are physical faults, software faults are design faults, which are very difficult to visualize, detect, and correct. Therefore, software reliability is a much more difficult measure to obtain and analyze. It becomes essential to estimate, predict, and measure software reliability

IV. Technical Aspects Related To Software Reliability

4.1 Fault Prevention: To enhance the reliability of the software it is essential to prevent fault from occurrence. The general approaches that could be followed could be the interactive refinement of the user's/ system's requirements, the requirements engineering, good software design methods, structured programming, writing clear code, good documentation, right implementation, and perfect maintenance. These guidelines are the fundamental techniques that can prevent software faults.

Recently, formal methods and software reuse have been attempted to resolve the software quality problem and thus as tool to prevent faults. Formal methods are a particular kind of mathematically based techniques for the specification, development and verification of software systems. Software reuse encourages to write effective modular code more economically which can be redesigned, and tested as and when needed. This also decreases the complexity of the software system as a whole. Error/fault detection and correction are simplified. Prototypes are easier to construct, evaluate and synthesize into the product up to the customer satisfaction level. This is why object- oriented paradigms and techniques are receiving much attention nowadays. This approach is proving one of the important technique in fault prevention.

4.2 Fault removal: When formal methods are in full swing, formal design proofs might be available to achieve mathematical proof of correctness for programs. Also, fault-monitoring assertion could be employed through executable specification, and test cases could be automatically generated to achieve efficient software verification.

However, before this happens, practitioners will have to rely mostly on software testing techniques to remove existing faults. Microsoft, for example, allocates as many software testers as software developers, and employs a buddy system which binds the developer of every software component to its tester for their daily work (Lyu,1995). The key question to reliability engineers, then, is how to derive testing - quality measures (e.g., test-coverage factors) and establish their relationships to reliability.

Another practical fault removal schemes might be formal inspection, walkthroughs, and reviews. These have found to be very tactical in finding faults, correcting, and verifying the corrections. These are carried out by a group of peers with a vested interest in the work product during pretest phases of the life cycle.

4.3 Fault tolerance: Fault tolerance is the survival attribute of a system. It gives the software the ability to deliver continuous service to their uses even in the conditions of error/fault. This feature, also might be called robustness of the software, gives software a high level of reliability because the software would not stop functioning even in the conditions of fault encountered and would be able to deliver its services on continuous basis. These software faults may or may not manifest themselves during system operations, but when they do, software fault tolerance techniques should provide the necessary mechanisms to the software system to prevent system failure from occurring.

4.4 Fault/failure forecasting: It would be an important technique to minimize failure and thus increasing the reliability of the software system. Fault/failure forecasting involves formulation of the fault-failure relationship, the collection of failure data, an understanding of the operational environment, the establishment of reliability models, the application and selection of reliability models by tools, the analysis and interpretation of results. The early forecasting would be an aid to plan, organize, and design the risk of occurring faults and failure and according manage to mitigate the impact of failure to make the system tolerant to fault./failure, thereby increasing the level of reliability. This would certainly reduce to cost to meet in failure in case it becomes a reality.

V. Software Reliability Measurement

Absolute measurement of software reliability is hard to achieve, yet number of scientists have proposed so many number of models to measure reliability of software and claim for the same. Measurement of software reliability includes two fundamental activities: Estimation and Prediction.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 12, December 2016)

Estimation: This activity determines current software reliability status from data obtained from past software. These data are mostly failure data obtained during system testing and operation. These data are used for statistical inferences regarding software reliability. Its main objective is to assess current reliability level.

Prediction: To plan for and maintain the high level of reliability for a future software Prediction of software reliability is an important activity. Prediction could be done from the data available from software metrics (project, product, and process metrics) and measures. Prediction can be based on two situations:

a. Availability of failure data: When the software remains in the stage of system testing or operation stage, the estimation techniques can be used to parameterize and verify software reliability models, which can perform future reliability prediction.

b. Non-Availability of failure data: Particularly in the design or coding phase of software development process the failure such cases metrics obtained from the software development process and the characteristics of the resulting product can be used to determine reliability of the software.

VI. Software Reliability Prediction

Various techniques have been used to predict software reliability; here authors have studied three recent techniques : 1. Using Fuzzy Logic, 2. Using Artificial Neural Network, and 3. Using Genetic Algorithm.

6.1 Using Fuzzy Logic: Fuzzy logic deals with the kind of uncertainty that is inherently human in nature. It is based on "degrees of truth" rather than usual "true or false" binary states. It is a method of reasoning that resembles human reasoning. This technique, which uses the mathematical theory of fuzzy sets, simulates the process of normal human reasoning by allowing the computer to behave less precisely and logically than conventional computer methods require. It deals with reasoning which is approximate rather than precisely deducted from classical predicate logic. The thinking behind this approach is that decision making is not always a matter of black and white or true or false; it often involves gray areas, that is, *may be*. Many situations are not 100% true or false. There are many decision making problems that do not fit strictly into the true/false situation that require mathematical models for solution. It can be thought of as the application side of fuzzy set theory dealing with well thought out real world expert values for a complex problem Fuzzy logic was initiated in 1963 by L.A. Zadeh, Professor of Computer Science.

Fuzzy logic based software reliability model was first presented by Karunanithi, *et.al*.(Padala & Mohan, 2016).

Fuzzy logic is proven to be capable of modeling highly nonlinear and multidimensional models. Fuzziness refers to non-statistical imprecision and vagueness in information and data. The difference between fuzzy logic and probabilistic logic consist in the fact that the fuzzy logic uses truth degrees as a mathematical model for vague facts while the probabilistic one is mathematical model for random facts. The linguistic values are used for writing the If - Then rules. Researchers in this area have felt that fuzzy logic is vital for Software reliability prediction. Yuan *et. al.* in used fuzzy subtractive clustering integrated with module order modeling for software quality prediction. First Fuzzy Subtractive clustering is used to predict the number of faults then module order modeling is used to predict whether modules are fault prone or not. Xu *et. al.*(2000) introduced the fuzzy nonlinear regression (FNR) modeling technique as a method for predicting fault ranges in software models, Kumar & Jayram,2014, Padala & Mohan, 2016).

Fuzzy logic is difficult to apply when people supply the membership information. The problems stem from the linguistic vagueness to difficulties in supplying the definitions needed. One area in which fuzzy logic is being used extensively is in consumer products where the input is provided by sensors rather than by people. Fuzzy logic provides smooth motion in consumer products, in the area of controls, in predicting accident risk, and many others.

6.2 Using Artificial Neural Networks: Many factors like software development process, and software test or use characteristics, software complexity, non-algorithmic problems, and nature of software faults and the possibility of occurrence of failure affect the software reliability behavior. Neural network (NN) methods normally approximate any non-linear continuous function. A neural network is configured for a specific application, such as data classification or pattern recognition, through a learning process. Just as in biological systems, learning involves adjustments to the synaptic connections that exist between the neurons. NN can differ on – the way their neurons are connected; the specific kinds of computations their neurons do; the way they transmit patterns of activity throughout the network; and the way they learn including their learning rate. NN are being applied to increasingly large number of real world problems. It can solve problems that are too complex for conventional technologies – problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be defined. So more attention is given to neural network based methods now -a-days.

Neural network based software reliability model was first presented by Karunanithi *et. al.*(1991) to predict cumulative number of failures. They consider execution time as the input of the neural network. In their approach, they used different networks like Feed Forward neural networks. Recurrent neural network like Jordan neural network and Eleman neural network. Two different training regimes like Prediction and Generalization are also used in their study. They compared their results with some statistical models and found better prediction than those models. Karunanithi et al. also used connectionist models for software reliability prediction. They applied the Falman's cascade Correlation algorithm to find out the architecture of the neural network. They applied the Falman's cascade Correlation algorithm to find out the architecture of the neural network. They considered the minimum number of training points as three and calculated the average error (AE) for both end point and next -step prediction. Their results concluded that the connectionist approach is better for end point prediction. (Kumar & Jayram,2014). Sitte(1999) presented a neural network based method for software reliability prediction. He compared the approach with recalibration for parametric models using some meaningful predictive measures with same datasets. They concluded that neural network approach is better predictors, (Bisi & Goyal,2012).

Cai *et. al.* proposed a neural network based method for software reliability prediction. They used back propagation algorithm for training. They evaluated the performance of the approach by varying the number of inputs nodes and number of hidden nodes. They concluded that the effectiveness of the approach generally depends upon the nature of the handled data sets. Tian and Noore proposed an on-line adaptive software reliability prediction model using evolutionary connectionist approach based on multiple-delayed -input single -output architecture.

6.3 Using Genetic Algorithm: Genetic Algorithms (GAs) were developed by Prof. John Holland, *et. al.* at the University of Michigan during the 1960s and 1970s. It is an iterative procedure that represents its candidate solutions as strings of genes called chromosomes and measures their viability with a fitness function. The fitness function is a measure of the objective to be obtained (maximum or minimum). As in biological systems, candidate solutions combine to produce offspring in each algorithmic iteration called a generation. The offspring themselves can become candidate solutions. From the generation of parents and children, a set of the fittest survive to become parents that produce offspring in the next generation.

Offspring are produced by specific genetic operators that include reproduction, crossover, and mutation. The three most important aspects of using genetic algorithm are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that one can try many different variations to improve performance. Find multiple optima (species - if they exist), or parallelize the algorithms Genetic algorithms are machine learning and optimization schemes, much like neural networks. However, genetic algorithms do not appear to suffer from local machine as badly as neural networks do. Genetic algorithm is based on the model of evolution, in which a population evolves towards overall fitness, even though individual perish. Evolution dictates that superior individuals have a better chance of reproducing to inferior individuals, and thus are more likely to pass their superior traits on to the next generation. This "Survival of the fittest" criterion was first converted to an optimization algorithm by Holland in 1975, and is today a major optimization technique for complex, nonlinear problems. Oliveira et al. proposed the using of genetic programming (GP) to obtain software reliability model for forecasting the reliability and extended this work by boosting the GP algorithm using re-weighting. The re-weighting algorithm calls many times the learning algorithm with assigned weights to each example. Each time, the weights are computed according to the error (or loss) on each example in the learning algorithm. In this way, the learning algorithm is manipulated to look closer at examples with bad prediction functions. Sheta uses genetic algorithms to estimate the COCOMO model parameters for NASA Software Projects. The same idea is implemented for estimating the parameters of different SRGM models using PSO (Kumar & Jayram,2014). Y. Zhang *et al.* (2006) anticipated MTBF disappointment information arrangement of software reliability by genetic programming algorithm. The transformative model of GP was then tacked and assessed by attribute criteria for some routinely used programming testing cases,(Lohmor & Sagar,2016). Genetic algorithms provide a set of efficient, domain-independent search heuristics for a broad spectrum of applications that include – Dynamic process control, Induction of optimization of rules, discovering new connectivity topologies, simulating biological models of behavior and evolution, pattern recognition, scheduling, parallel processing, and many more.

VII. CONCLUSION

Today we expect for a zero-defect software but the problem of reliability in software systems is a well- known fact. There is hardly an approach that will solve this problem, instead the solution will be the combination of several approaches. Improvements are needed throughout the whole life cycle of a software. These improvements include, for example, specification and design, coding and testing, verification and validation, certification, management of change as well as maintenance. The Cleanroom methodology takes into account the methods for specification and design, verification and validation, as well as certification. Various approaches – Formal methods, agile technology and others are becoming popular these days besides Cleanroom methodology which supports the idea and philosophy that it is possible to develop zero - defect software.

It can be concluded that the statistical quality control of software products is an important issue. The certification process is central in this effort. This process is highly dependent on relevant software reliability models and a sound basis for prediction and estimation. The basis includes relevant failure data, i.e. data which is obtained under the circumstances of fulfilling the assumptions of the reliability models. This means that the failure data during testing and other type of analysis must match to the failure data encountered during operation.

REFERENCES

[1] Adnan, W, A., and Mashkuri, 2014. "An Integrated Neural Fuzzy System of Software Reliability Prediction", IEEE Software. IJIRAE, Vol.1 Issue 4.

[2] Ando, T., H. Okumara and Dohi,T.,2006. "Software Reliability Prediction using Neural Network with Encoded Input", International Journal of Computer Applications (0975 – 8887) Volume 47– No.22.

[3] Bisi,M. and Goyal,N.K.,2012. " Estimating Markov Modulated Software Reliability Models via EM Algorithm[C]". Proceedings of the 2nd IEEE International Symposium on Dependable, Automatic and Secure Computing.

[4] Cheung, L.,Roshandel, R., Medvidovic, N. and Golubchik,L. 2008. "Early Prediction of Software Component Reliability", ACM, pp. 111-121.

[5] Constantinescu. N, Iancu,I., 2008. "Fuzzy Identity Authentication, Latest Trends on Computers", 1(1), pp 168-173.

[6] Farr, W. "Software reliability modeling survey ", 1996. Naval Surface Warfare Center, Technical Foundation, pp.73-109.

[7] Jelinski, Z. and Moranda, P.B.,1972. "Software reliability research, In Statistical Computer Performance Evaluation", (Edited by W. Freiberger), Academic Press, New York, pp. 465-484.

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

[9] K. Y. Cai, C.Y.Wen, and Zhang, M.L.,1991. "A critical review on software reliability modeling.", Reliability Engineering and System Safety 32(3), 357-371.

[10] Karunanithi. N., Malaiya,Y.K. and Whitley, D.,1991. "Prediction of software reliability using neural networks", Proceedings of the Second IEEE International Symposium on Software Reliability Engineering, pp.124-130,

[11] Karunanithi, N., Malaiya,Y.K. and Whitley, D.,1992. "Prediction of software reliability using connectionist models", IEEE Transactions on Software Engineering, Vol. 18, no. 7, pp. 563-574.

[12] Karunanithi. N., Whitley, D.and Malaiya, Y.K.1992 "Using neural networks in reliability prediction", IEEE Software, vol. 9, no.4, pp. 53-59.

[13] Ke-han, ZHANG., Li Ai-guo and SONG Bao-Wei,2008. "Estimating Parameters of Software Reliability Models using PSO", Computer Engineering and Applications, 44(11):47-49.

[14] Kimura, M, Yamada,S. and Osaki,S.2011. "Statistical Software reliability prediction and its applicability based on mean time between failures", IJRC, Vol. 8, Issue 3, No-2.

[15] Klir, George J., St Clair, H. Ute and Yuan, 2014. "Fuzzy set theory: foundations and applications", IJIRAE, Vol., 1 Issue May-2014.

[16] Lohmor, S. and Sagar,B.B. 2016. "A Comprehensive Review on Software Reliability Growth Models utilizing Soft Computing Approaches", Intelligent Systems Technologies and Applications 2016, Advances in Intelligent Systems and Computing 530, DOI 10.1007/978-3-319-47952-1_40,Springer Int. Pub.

[17] Lyu,M.R.,1992. "Applying Reliability Model More Effectively", University of Lowa, Jet Prolusion Laboratory. Caltech, July 1992, pp. 43-52.

[18] Lyu, M. R.,1995. "Handbook of Software Reliability Engineering", McGraw-Hill Publishing, ISBN 0-07-039400-8.

[19] Minohara, T. and Tohma,Y. 1995 "Parameter Estimation of Hyper - Geometric Distribution Software Reliability Growth Model by Genetic Algorithms[C]", Proceedings of the 6th IEEE Int. Symp. on Software Reliability Engg. (ISSRE 1995),Toulouse, France, pp.324-329.

[20] Musa, J. D., 1975. "A Theory of Software Reliability and its Applications", IEEE Transactions on Software Engineering, Vol. SE -1, No.3, pp.312-327.

[21] Musa, J.D. and Okumoto, K.,1984. "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement", 7th Int'l Conf. on Soft. Engg. (ICSE), IEEE, pp.230-238.

[22] Musa, J.D., Iannino, A. and Komodo, K., 1987."Software Reliability: Measurement, Prediction and, Application", McGraw-Hill Pub.

[23] Ohba, M., 1984. "Software Reliability-Analysis Models", IBM Journal of Research and Development, Vol. 28, No. 4, pp.428-443.

[24] Oliveira, E. O., Aurora Pozo, and Silvia Regina Vergilio, 2006. "Using boosting techniques to im- prove software reliability models based on genetic programming.", Tools with Artificial Intelligence, IEEE International Conference, ICTAI, pp. (653-650).

[25] Padala,Lakshman Rao and Mohan,E., 2016. "Fuzzy Analysis for the Cost Effective Software Evolution Exertion Appreciation", International Journal & Magazine for Engineering, Technology, Management, and Research, Vol. 3, No. 7, pp.462-466.

[26] Schick, G. J. and Wolverton, R.W.,1978. "An Analysis of computing software reliability models", IEEE Trans. Software Eng., Vol. SE-4, pp.104-120,

[27] Sitte,R.,1999. "Comparison of software – reliability - growth predictions: neural networks vs parametric recalibration",IEEE Transactions on Reliability, vol. 48, no. 3, pp. 285- 291.

[28] Shooman, M .L, 1987 "Yes, Software Reliability Can be Measured and Predicted", Division of Computer Science, Polytechnic University, pp.121-122.

[29] Tian, J.,2002. "Better Reliability Assessment an Prediction through Data Clustering", IEEE Transactions on Software Engineering,Vol. 28, No. 10.

[30] Tian,L. and Noore,A. 2005. "Evolutionary neural network modeling for software cumulative failure time prediction", Reliability Engineering and System Safety 87,45-51.

[31] Tian,L. and Noore,A.2005. "On-line prediction of software reliability using an evolutionary connectionist model", The Journal of Systems and Software, 77,173-180.

[32] Turban, Efraim and Aronson, Jay E.,2005. "Decision Support Systems and Intelligent Systems",6th Ed., pp.673.

[33] Vijaya kumar, H.S. and Jayram,M.A.,2014. "On Applications of Soft Computing Assisted Analysis for Software Reliability", Int. J. of Innovative Research in Advanced Engg., vol.1, Issue 4.

[34] Viswanath, S.P.K.,2014. "Software Reliability Prediction using Neural Networks", PhD. Thesis, Indian Institute of Technology, Kharagpur. IJIRAE, Vol. 1, Issue 4.

[35] Wason, Ritika, Ahmed, P. and Qasim Rafiq, M.,2012 "New Paradigm for Software Reliability Estimation", International Journal of Computer Applications (0975-887) Vol. 44, No.14.

[36] Xu, Z. and Allen, E.B.,2000. "Prediction of Software Faults Using Fuzzy Nonlinear Regression Modelling", IEEE Software, 281-290.

[37] Yadav, A. and Khan,R.A.2009. "Critical Review on Software Reliability Models" International Journal of Recent Trends in Engineering, Vol. 2, No.3.

[38] Yamada, S., Ohba,M. and Osaki,S.1983. "S- Shaped Reliability Growth Modeling for Error Detection", IEEE Transaction Reliability, pp. 475-478.

[39] Yuan, X., K. Ganesan "An application of Fuzzy clustering to software quality prediction", IEEE Software, 2000.

[40] Zadeh, L. A. " Fuzzy Sets, Information and Control", 1965.

[41] Zadeh, L. A. "Fuzzy algorithms, Info, & Ctl ", Vol. 12, 1968.

[42] Zhang,Y. and Huashan,C.,2006. "Predicting for MTBF failure data series of software reliability by genetic programming algorithm." Intelligent Systems Design and Applications, ISDA'06. Sixth International Conference, Vol. 1, pp. (666-670).