International Journal of Computer Applications Technology and Research
Volume 3– Issue 4, 267 - 272, 2014, ISSN: 2319–8656

A Review on Parameter Estimation Techniques of Software Reliability Growth Models

Karambir Bidhan
University Institute of Engineering and Technology
Kurukshetra University Haryana,
India

Adima Awasthi
University Institute of Engineering and Technology
Kurukshetra University Haryana,
India

Abstract: Software reliability is considered as a quantifiable metric, which is defined as the probability of a software to operate without failure for a specified period of time in a specific environment. Various software reliability growth models have been proposed to predict the reliability of a software. These models help vendors to predict the behaviour of the software before shipment. The reliability is predicted by estimating the parameters of the software reliability growth models. But the model parameters are generally in nonlinear relationships which creates many problems in finding the optimal parameters using traditional techniques like Maximum Likelihood and least Square Estimation. Various stochastic search algorithms have been introduced which have made the task of parameter estimation, more reliable and computationally easier. Parameter estimation of NHPP based reliability models, using MLE and using an evolutionary search algorithm called Particle Swarm Optimization, has been explored in the paper.

Keywords: Software Reliability, Software Reliability Growth Models, Parameter Estimation, Maximum Likelihood Estimation, Partical Swarm Optimization

1. INTRODUCTION

Software can be defined as an instrument comprising a set of coded statements which takes a discrete set of inputs and transform them into a discrete set of outputs. Software may contain discrepancies or faults which may result into software failures. Any kind of deviation from the desired behaviour of the software is apparently unwanted for the user and to check the correctness of the program two approaches were used which can be named as : program proving and program testing. Program proving is a formal and mathematical approach in which a finite sequence of logical statements ending in the statement, usually the output specification statement, to be proved is constructed. Each of the logical statements is either an axiom or is a statement derived from earlier statements by the application of an inference rule. However Gerhart and Yelowitz [1] presented various programs which were proved to be correct by this approach but in actually, were containing faults. On the other hand program testing is more practical approach and is heuristic in nature. Program testing basically involves symbolic or physical execution of a set of test cases with the objective of exposing embedded faults in the program. Program testing, like that of program proving is an imperfect tool for ensuring program correctness. As these approaches cannot completely assure about the correctness of the program, a metric is required which can help in assessing the degree of program correctness and software reliability fulfils this requirement. Reliability of a software can be defined as: "*The probability of failure-free software operation for a specific period of time in a specified environment*" [2]. Reliability of a software needs to be assessed before it is delivered to the customer. Various software reliability growth models (SRGM) has been introduced for predicting the reliability of a software. Although according to an observation made by M. R. Lyu in [3], "There is no universally acceptable model that can be trusted to give accurate results in all circumstances; users should not trust claims to the contrary." Every model has some advantages and some disadvantages. The choice regarding which model to follow for the prediction depends upon the requirements of the software.

We ask that authors follow some simple guidelines. This document is a template. An electronic copy can be downloaded from the journal website. For questions on paper guidelines, please contact the conference publications committee as indicated on the conference website. Information about final paper submission is available from the conference website

2. SOFTWARE RELIABILITY GROWTH MODELS

A Software Reliability Growth Model can be considered as one of the fundamental techniques to assess the reliability of a software quantitatively. Any software reliability model presents a mathematical function which illustrates defect detection rates. These models are classified in two categories: Concave and S-shaped models , which can be illustrated with the help of fig-1[4].

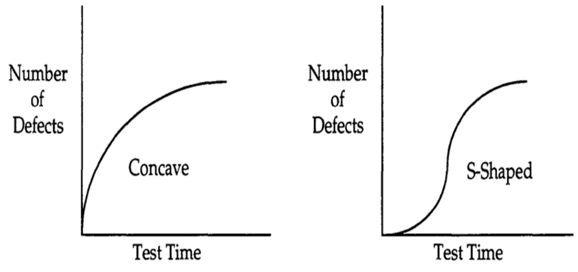

Figure-1: Concave and S-Shaped Models

Concave rnodels are so-called because they bend downward. S-shaped models, on the other hand, are first convex and then concave. This reflects their underlying assumption that early testing is not as efficient as later testing, so there is a "ramp-up" period during which the defect-detection rate increases. This period terminates at the inflection point in the S-shaped curve. The most important thing about both models is that they have the same asymptotic behavior: The defect-detection rate decreases as the number of defects detected increases, and the total number of defects detected asymptotically approaches a finite value. Several models had been proposed for assessing the reliability of the softwares, basic assumptions of some of the models are discussed below :

Goel-Okumoto Model :

This model was first introduced by Goel and Okumoto [5] and is based on the following

assumptions:

- All faults in the software are mutually independent from the failure detection point of view.
- The number of failures detected at any time is proportional to the current number of faults in the software. This means that the probability of the failures or faults actually occurring, i.e., being detected, is constant.
- The isolated faults are removed prior to future test occasions.
- Each time a software failure occurs, the software error which caused it is immediately removed and no new errors are introduced.

These assumptions lead to the following differential equations:

$$\frac{\partial}{\partial t}\,\mu(t)| = b(a - \mu(t)) \qquad (1)$$

In this equation, a represents the expected total number of faults in the software before testing. Parameter b stands for the failure detection rate or the failure intensity of a fault and μ (t) for the expected number of failures detected at time t. Solving above equation for μ (t), we obtain the following mean value function

$$\mu(t) = a(1 - e^{-bt}) \qquad (2)$$

Yamada S-Shaped Model :

The Yamada S-Shaped model was first introduced in Yamada et al. [6]. The model is based on the following assumptions:

- All faults in the software are mutually independent from the failure detection point of view.
- The probability of failure detection at any time is proportional to the current number of faults in the software.
- The proportionality of failure detection is constant.
- The initial error content of the software is a random variable.
- A software system is subject to failures at random times caused by errors present in the system.
- The time between the (i-1)th and the ith failure, depends on the time of the (i-1)th failure.

Musa's basic execution time model [7]

It assumes that there are N software faults at the start of testing, each is independent of others and is equally likely to cause a failure during testing. A detected fault is removed with certainty in a negligible time and no new faults are introduced during the debugging process. The hazard function for this model is given by

$$z(\tau) = \phi f(N - n_c) \qquad (3)$$

where τ is the execution time utilized in executing the program up to the present, f is the linear execution frequency (average instruction execution rate divided by the number of instructions in the program), ϕ is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, and n, is the number of faults corrected during (0,τ). One of the main features of this model is that it explicitly emphasizes the dependence of the hazard function on execution time. Musa also provides a systematic approach for converting the model so that it can be applicable for calendar time as well.

3. PARAMETER ESTIMATION TECHNIQUES

A software reliability model is simply a function and fitting this function to the data means estimating its parameters from the data. One approach to estimating parameters is to input the data directly into equations for the parameters. The most common method for this direct parameter estimation is the maximum likelihood technique. Maximum Likelihood Estimation is a method which is used for the estimation of the parameters of a statistical models. If maximum likelihood estimation is applied to a data set for a given statistical model then it provides the estimates of that model's parameters. A second approach is fitting the curve described by the function to the data and estimating the parameters from the best fit to the curve. The most common method for this indirect parameter estimation is the least squares technique. Although methods like MLE and LSE provide a way to estimate the parameters of reliability models, but the model parameters are normally in nonlinear relationships and this makes traditional parameter estimation techniques suffer many problems in finding the optimal parameters to tune the model for a better prediction. Parameter estimation problem for nonlinear systems can be stated and formulated as a function optimization problem in which the objective is to obtain a set of parameters that provide the best fit to a measured data based on a specific type of function to be optimized. Such parameters are obtained using a search technique in the space of values specified in advance. Searching techniques are bound to the complexity of the search space, and the use of Gradient search might find local minimum solution but not optimal ones. Stochastic search algorithms, on the other hand, such as Evolutionary Algorithms e.g Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) present a more reliable functionality in estimating models' parameters.

4. LITERATURE REVIEW

4.1 Parameter Estimation Using Maximum Likelihood Estimation

Barnard and Bayes [8] depicted that for most mathematical models, if the number of parameters is large and the observed data is erroneous, calibration can be performed in maximum likelihood (ML) framework, where the state estimate is the parameter which maximizes the likelihood function.

Knafl [9] proposed existence conditions for maximum likelihood parameter estimates for several commonly employed two-parameter software reliability models. For these models, the maximum likelihood equations were expressed in terms of a single equation in one unknown.

Bounds were given on solutions to these single equations problems to serve as initial intervals for search algorithms like bisection. Uniqueness of the solutions is established in some cases. Results were given for the case of grouped failure data.

Okumoto [10] reviewed four analytical software reliability models which are used for estimating and monitoring software reliability. These models include Times Between failure Models, Failure Count Models, Fault Studying models and Input Domain Models. Goel and Okumoto proposed Non homogeneous Poisson Process model which lie in the category of Failure Count models of software reliability estimation. In this model it is assumed that failures occur during execution of the software, at random times because of faults present in the software. If we represent cumulative number of failures occurred in system till time t by N(t) then the non homogeneous poisson model can be represented as follows:

$$(P\{N(t = y)\}) = \frac{(m(t))^y}{y!} e^{-m(t)} \qquad (4)$$

$$\text{Where } m(t) = a(1-e^{-bt}) \qquad (5)$$

$$\lambda(t) = m'(t) = abe^{-bt} \qquad (6)$$

m(t) in above equations is the expected no. of failures observed by time t and λ(t) is the failure intensity function. Here 'a' is the expected number of failures to be observed eventually and 'b' is the fault detection rate per fault.

Knafl and Morgan [11] proposed that the reliability of the software can be estimated using software reliability growth models, or a non-homogeneous poisson process model with mean value function μ(t). As in Goel-Gkumoto model μ(t) consists of two parameters a and b and in order to predict the reliability of any software the values of these parameters need to be estimated. These parameters can be estimated by using the Maximum Likelihood Estimation method. An observation interval of $(0, t_k]$ was considered and divided into various subintervals given by $(0, t_1]$, $(t_1, t_2]$……, $(t_{k-1}, t_k]$,the number of failures per subinterval is denoted by n_i where (i=1,2,3….k) with respect to the number of failures in $(t_{i-1}, t_i]$. Thus the likelihood function for mean value function μ(t) of G-D model can be given by—

$$L(n_1, \dots\dots, n_k) = \prod_{i=1}^{k} \frac{\{\mu(t_i)-\mu t_{i-1}\}^{n_i}}{n_i!} \exp\{-\mu(t_i) - \mu t_{i-1}\} \qquad (7)$$

By taking natural logarithm on both sides of equation (7)-

$$\ln L(n_1, \dots\dots, n_k) = \ln \prod_{i=1}^{k} \frac{\{\mu(t_i)-\mu t_{i-1}\}^{n_i}}{n_i!} \exp\{-\mu(t_i) - \mu t_{i-1}\} \qquad (8)$$

$$= \ln \sum_{i=1}^{k} \frac{\{\mu(t_i)-\mu t_{i-1}\}^{n_i}}{n_i!} \exp\{-\mu(t_i) - \mu t_{i-1}\} \qquad (9)$$

$$= \sum_{i1}^{k}\{ n_i \ln[\mu(t_i) - \mu(t_{i-1})] - [\mu(t_i) - \mu(t_{i-1})] - \ln n_i! \qquad (10)$$

Maximum likelihood estimates of the parameters a and b can be computed by taking the partial derivative of equation (8) with respect to each model parameter and then equating the derivatives to zero one by one. Thus the estimates of the model parameters will be computed as follows:

$$= \sum_{i=1}^{k}\{\frac{n_i}{a} + e^{-bt_i} - e^{-bt_{i-1}}\} = 0 \qquad (11)$$

$$\frac{\partial \ln L}{\partial b} = \sum_{i=1}^{k}\left(\frac{n_i}{e^{-bt_i} - e^{-bt_{i-1}}} - a\right)(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}) = 0 \qquad (12)$$

Expression for a and b can be obtained by solving the following equations:

$$a = \frac{\sum_{i=1}^{k} n_i}{1-e^{-bt_k}} \qquad (13)$$

$$\sum_{i=1}^{k}\left(\frac{n_i}{e^{-bt_i} - e^{-bt_{i-1}}} - \frac{\sum_{i=1}^{k} n_i}{1-e^{-bt_k}}\right)(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}}) = 0 \qquad (14)$$

Equation (12) is a nonlinear equation and its not possible to solve it analytically hence it must be solved numerically. It can be solved using newton's method. A modification of G-O model was presented by Hossain and Dahiya [12] in which Maximum likelihood (ML) equations were investigated and a sufficient condition for the ML estimators to be finite was given. In the G-O model the probability distribution function (p.d.f) of the time to first failure is given by:

$$g(t) = \frac{a\,b\,e^{-bt} e^{ae^{-bt}}}{e^{a}} \qquad (15)$$

this is an improper p.d.f which is a big drawback of this model. The modified model uses the proper p.d.f which is given as:

$$f(t) = \frac{a\,b\,e^{-bt} e^{ae^{-bt}}}{e^{a} - 1} \qquad (16)$$

using this equation the modified model was given by:

$$f(t) = \frac{a\,b\,e^{-bt} e^{ae^{-bt}}}{e^{a} - c} \qquad (16)$$

with the hope that it will do better than the G-O model.The model (5), when c=0, is G-O model and when c= 1, the corresponding pdf{time to failure} is proper. In this kind of model we might anticipate that m(∞) is finite. But when **c** = 1, then m(∞) = ∞ , giving rise to a new problem in determining the mean total number of failures in the system. So to avoid this situation the modified model needs to choose a {c: 0≤c<1 } that gives a better (in some sense) estimated mean number of failures in the system than the G-O estimate. H-D model is superior to the G-O model because: **1)** it is more flexible; 2) it assigns less weight at infinity in the pdf of the time to failure; and 3) the sufficient condition for the existence of finite solution of ML equations is the same as the necessary & sufficient condition for the GO model.

4.2 Parameter Estimation Using particle Swarm Optimization

Kennedy and Eberhart [13] depicted that PSO is a simple model of social learning whose emergent behaviour has found popularity in solving difficult optimization problems. The initial metaphor had two cognitive aspects, individual learning and learning from a social group. Where an individual finds itself in a problem space by using its own experience and that of its peers to move itself toward the solution

$$v_{t+1} = v_i + \varphi_1\beta_1(p_i - x_i) + \varphi_2\beta_2(p_g - x_i) \qquad (18)$$

$$x_{t+1} = x_t + v_{t+1} \qquad (19)$$

where constants φ_1 and φ_2 determine the balance between the influence of the individual's knowledge (φ_1) and that of the group (φ_2) (both set initially to 2), β_1 and β_2 are uniformly distributed random numbers defined by some upper limit, β_{max}, that is a parameter of the algorithm, p_i and p_g are the individual's previous best position and the group's previous best position, and x_i is the current position in the dimension considered.

This was found to suffer from instability caused by particles accelerating out of the solution space. Eberhart et al [14] proposed clamping scheme that limited the speed of each particle to a range [-v_{max}, v_{max}] with v_{max} usually being somewhere between 0.1 and 1.0 times the maximum position of the particle. This reduced the possibility of particles flying out of the problem space.

The most problematic characteristic of PSO is its propensity to converge, prematurely, on early best solutions. Many strategies have been developed in attempts to overcome this but by far the most popular are inertia and constriction. Therefore Shi and Eberhart [15] introduced the term inertia, w, as follows:

$$v_{t+1} = \omega\, v_i + \varphi_1\beta_1(p_i - x_i) + \varphi_2\beta_2(p_g - x_i) \qquad (20)$$

Eberhart and Shi [16] introduced an optimal strategy of initially setting ω to 0.9 and reducing it linearly to 0.4, allowing initial exploration followed by acceleration toward an improved global optimum. They also showed that combining them by setting the inertia weight, ω, to the constriction factor, χ , improved performance across a wide range of problems.

Clerc and Kennedy [17] proposed an idea of introducing constriction, χ , which alleviated the requirement to clamp the velocity and is applied as follows:

$$\chi = \frac{2}{2-\varphi-\sqrt{\varphi^2-4\varphi}} \qquad (21)$$

Where $\varphi = \varphi_1 + \varphi_2$, $\varphi > 4$

Kennedy [18] revisited the constricted PSO and examined whether the added components were necessary, and whether any further components could be removed. Various experiments were performed with a view to paring the process for further efficiency gains. To achieve this, a Gaussian PSO was developed. In this implementation the entire velocity vector is replaced by a random number generated around the mean (p_{id} + p_{gd})/2 with a Gaussian distribution of |p_{id}_p_{gd}| in each dimension (d). This effectively means that the particles no longer 'fly' but are 'teleported'. Kennedy justified this departure on the grounds that it is the social aspect of the swarm that is more important to its effectiveness. This was empirically backed up with the Gaussian influenced swarms performing competitively.

Merwe et al [19] investigated the application of the PSO to cluster data vectors. Two algorithms were tested, namely a standard gbest PSO and a Hybrid approach where the individuals of the swarm were seeded by the result of the K-means algorithm. The two PSO approaches were compared against K-means clustering, which showed that the PSO approaches have better convergence to lower quantization errors, and in general, larger inter-cluster distances and smaller intracluster distances. It was shown that how PSO should be used to find the centroids of a user specified number of clusters. The algorithm was then extended to use K-means clustering to seed the initial swarm. The second algorithm basically used PSO to refine the clusters formed by K-means. The new PSO algorithms was evaluated on six data sets, and compared to the performance of K-means clustering. Results showed that both PSO clustering techniques have much potential.

Monson and Seppi [20], showed that particle motion could be further improved through changing the mechanism to a system influenced by Kalman filtering in which the motion was entirely described by predictions produced by the filter. Once again, the justification for such a radical change to particle movement was the maintenance of the social aspect of PSO, which was achieved through the Kalman filter's sensor model. The approach, whilst providing very accurate optimization, was found to be computationally expensive compared with the canonical PSO.

Parsopoulos and Vrahatis [21] modified the constricted algorithm to harness the explorative behaviour of global search and exploitative nature of a local neighbourhood scheme. To combine the two, two velocity updates were initially calculated:

$$G_{t+1} = \chi\{v_i + \varphi_1\beta_1(p_i - x_i) + \varphi_2\beta_2(p_g - x_i)\} \qquad (22)$$
$$L_{t+1} = \chi\{v_i + \varphi_1\beta_1'(p_i - x_i) + \varphi_2\beta_2'(p_{gl} - x_i)\} \qquad (23)$$

where G and L are the global and local velocity updates respectively, p_g is the global best particle position and p_{gl} is the particle's local neighborhood best particle position. These two updates were then combined to form a unified velocity update (U), which is then applied to the current position:

$$U_{t+1} = (1-u)L_{t+1} + uG_{t+1} \quad u \in [0,1] \qquad (24)$$
$$x_{t+1} = x_t + U_{t+1} \qquad (25)$$

where u is a unification factor that balances the global and local aspects of the search and suggestions were given to add mutation style influences to each in turn.

Kaewkamnerdpong and Bentley [22] introduced more realistic model of particle perception to PSO, with the aim of making it a viable option for applications that might otherwise suffer from imperfect communication, such as robotics. In the natural world, a social animal would often suffer imperfect perception of the other animals in its sociometric neighbourhood and must rely on information it receives via limited perceptive acuity. To model this, the particles were not allowed to communicate directly but only to observe particles within their pre-set perceptive range (replicating the stigmergic behaviour of some species, such as dancing honeybees); this was also realistic in that the information was regarded as more reliable where its source was closer. Each particle was also allowed to observe the local area (i.e. sample the local solution space). If there was a better solution nearby it used that as its individual best position. The position update algorithm for this method is dependent on the current state of the particle and can be summarised as follows:

• If there is an observed position $p_{observed} > p_i$ then set p_i to $p_{observed}$.

• If no neighbouring particles are perceived then do not use a social component.

• If multiple particles are perceived use their average current position as the group

best(p_g).

• If a single particle is perceived then use its position as the group best (p_g).

Habibi et al. [23] developed a hybrid PSO, with Ant Colony (AC) and Simulated Annealing (SA). The AC algorithm replaced the individual best element of PSO, whilst the cooling process of SA was used to control the exploration of the group best element, both of which were then applied within the PSO framework (all random numbers being generated using a Gaussian distribution function). Experimentation with known TSP instances indicated that the hybrid approach was capable of finding good approximations efficiently.

Sheta [24] In this paper proposed the preliminary idea of using particle swarm optimization (PSO) technique to help in solving the reliability growth modeling problem had been explored. The proposed approach was used to estimate the parameters of the well known reliability growth models such as the exponential model, power model and S-shaped models. The results were promising.

According to Zhu [25] *et.al* Particle swarm optimization (PSO), a swarm intelligence algorithm, had been successfully applied to many engineering optimization problems and shown its high search speed in these applications. However, as the dimension and the number of local optima of optimization problems increase, PSO and most existing improved PSO algorithms such as, the standard particle swarm optimization (SPSO) and the Gaussian particle swarm optimization (GPSO), were easily trapped in local optima. In this paper a novel algorithm was proposed which was based on SPSO called Euclidean particle swarm optimization (EPSO) which had greatly improved the ability of escaping from local optima. To confirm the effectiveness of EPSO, five benchmark functions had been employed to examine it, and compared it with SPSO and GPSO. The experiments results showed that EPSO is significantly better than SPSO and GPSO, especially obvious in higher-dimension problems.

Engelbrecht [26] proposed a heterogeneous PSO (HPSO) in this paper, where particles were allowed to follow different search behaviors selected from a behaviour pool, thereby efficiently addressing the exploration–exploitation trade-off problem. A preliminary empirical analysis showed that using heterogeneous swarms would give better results.

According to Quin [27] software reliability prediction classifies software modules as fault-prone modules and less fault-prone modules at the early age of software development. As to a difficult problem of choosing parameters for Support Vector Machine (SVM), this paper introduced Particle Swarm Optimization (PSO) to automatically optimize the parameters of SVM, and constructed a software reliability prediction model based on PSO and SVM. Finally, the paper introduced Principal Component Analysis (PCA) method to reduce the dimension of experimental data, and inputs these reduced data into software reliability prediction model to implement a simulation. The results showed that the proposed prediction model surpassed the traditional SVM in prediction performance.

According to Malhotra *et.al* [28] software quality includes many attributes including reliability of a software. Prediction of reliability of a software in early phases of software development would enable software practitioners in developing robust and fault tolerant systems. The purpose of the paper was to predict software reliability, by estimating the parameters of Software Reliability Growth Models (SRGMs). SRGMs are the mathematical models which generally reflect the properties of the process of fault detection during testing. Particle Swarm Optimization (PSO) has been applied to several optimization problems and has showed good performance. PSO is a popular machine learning algorithm under the category of Swarm Intelligence. PSO is an evolutionary algorithm like Genetic Algorithm (GA). In this paper the use of PSO algorithm was proposed to estimate parameters of delayed S-shaped model using machine learning method PSO and then compare the results with those of GA. The results are validated using data obtained from 16 projects. The results obtained from PSO had high predictive ability which was reflected by low error predictions. The results obtained using PSO are better than those obtained from GA.

Jadon et.al [29] proposed improved version of PSO called Self Adaptive Acceleration Factors in PSO (SAAFPSO) to balance between exploration and exploitation. The constant acceleration factors used in standard PSO had been converted into function of particle’s fitness. If a particle was more fit then it gave more importance to global best particle and less to itself to avoid local convergence. In later stages, particles would be more fitter so all would move towards global best particle, thus achieved the convergence speed. Experiment was performed and compared with standard PSO and Artificial bee colony (ABC) on 14 unbiased benchmark optimization functions and one real world engineering optimization problem (known as pressure vessel design) and results showed that proposed algorithm SAAFPSO dominated others.

According to Al gargoor et al [30], due to the growth in demand for software with high reliability and safety, software reliability prediction becomes more and more essential. Software reliability is a key part of software quality. However so many models had been proposed for the software reliability prediction but none of them can give accurate prediction for all cases. So in order to improve the accuracy of software reliability prediction the proposed model combine the software reliability models with the neural networks (NN). Particle swarm optimization (PSO) algorithm had been chosen and applied for learning process to select the best architecture of the neural network. The applicability of the proposed model was demonstrated through three software failure data sets. The results showed that the proposed model has good prediction capability and more applicable for software reliability prediction.

5. CONCLUSION

It has been explored that solving optimization problems using evolutionary algorithms like particle swarm optimization is found to be more reliable and computationally easier as compared to traditional techniques. Parameter estimation of the software reliability models is also an optimization problem. Traditionally MlE and LSE were used for the parameter estimation but evolutionary techniques like that of Genetic Algorithms, Swarm Intelligence etc have shown better results than that of the traditional methods. The model parameters usually follow nonlinear relationships which makes traditional parameter estimation techniques suffer many problems in finding the optimal parameters to tune the model for a better prediction these problems can be overcome by stochastic search methods.

6. ACKNOWLEDGMENTS

I would like to give my sincere gratitude to my guide Mr. Ajay Bidhan who encouraged and guided me throughout this paper.

7. REFERENCES

[1] S. Gerhart and L. Yelowitz, “Observations of availability in applications of modern programming methodologies”, IEEE transaction on Software Engineering ,pp 195-207, 1976.

[2] ANSI/IEEE, “Standard glossary of Software Engineering Terminology,” ANSI/IEEE, 1991.

[3] M. R. Lyu (Ed.), Handbook of Software Reliability Engineering, IEEE Computer Society Press, 1996.

[4] Wood, A., "Predicting software reliability," IEEE, vol.29, no.11, pp.69-77, 1996.

[5] A.L. Goel and K. Okumoto. Time-dependent error detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability, pp 206-211, 1979.

[6] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. IEEE Transactions on Reliability, pp 475-484, 1983.

[7] J. D. Musa, "A theory of software reliability and its application," IEEE Trans. Software vol. SE-1, 312-327, 1971.

[8] G. Barnard and T. Bayes, “Studies in the history of probability and statistics: IX. Thomas Bayes’s essay towards solving a problem in the doctrine of chances,” Biometrika, vol. 45, pp 293–315, 1985.

[9] Knafl, G.J., "Solving maximum likelihood equations for two-parameter software reliability models using grouped data," Third International Symposium on Software Reliability Engineering, pp 205-213, 1992.

[10] Goel, Amrit L. "Software reliability models: Assumptions, limitations, and applicability." IEEE Transactions on Software Engineering, pp 1411-1423, 1985.

[11] G.J. Knafl and J. Morgan, "Solving ML equations for 2-parameter Poisson-process models for ungrouped software- failure data", IEEE Transactions on eliability, pp 42-53, 1996

[12] Hossain, Syed A., and Ram C. Dahiya, "Estimating the parameters of a non-homogeneous Poisson-process model for software reliability.", IEEE Transactions on Reliability pp 604-612, 1993.

[13] Kennedy J, Eberhart RC,” Particle swarm optimization”, Proceedings of the IEEE international conference on neural networks, pp 1942–1948, 1995.

[14] Eberhart RC, Simpson P, Dobbins R ,“Computational intelligence PC tools” AP Professional, San Diego, CA, pp 212-226, 1996.

[15] Shi Y, Eberhart RC ,” A modified particle swarm optimizer” , Proceedings of the IEEE international conference on evolutionary computation. IEEE, pp 69–73, 1998.

[16] Eberhart RC, Shi Y , “Comparing inertia weights and constriction factors in particle swarm optimization” In: Proceedings of the IEEE congress evolutionary computation, San Diego, pp 84–88, 2000.

[17] Clerc M, Kennedy J ,”The particle swarm: explosion, stability and convergence in a multi-dimensional complex space”. IEEE pp 58–73, 2002.

[18] Kennedy J Bare bones ,”particle swarms “ In Proceedings of the IEEE swarm intelligence symposium Indianapolis, Indiana, USA, pp 80–87,2003

[19] Van der Merwe, D. W., and Andries Petrus Engelbrecht."Data clustering using particle swarm optimization.", Congress on Evolutionary Computation. Vol. 1. IEEE, 2003.

[20] Monson CK, Seppi KD,” The Kalman swarm” Proceedings of the genetic and evolutionary computation conference , Seattle, Washington,2004.

[21] Parsopoulos KE, Vrahatis MN “ a unified particle swarm optimization scheme”, Proceedings of international conference on computational methods in sciences and engineering, pp 868–873, 2004.

[22] Kaewkamnerdpong B, Bentley P,” Perceptive particle swarm optimization”, Proceedings of the seventh international conference on adaptive and natural computing algorithms , 2005.

[23] Habibi J, Zonouz SA, Saneei M,” A hybrid PS-based optimization algorithm for solving traveling salesman problem”, IEEE symposium on frontiers in networking with applications, pp 18–20 2006.

[24] Sheta, Alaa. "Reliability growth modeling for software fault detection using particle swarm optimization.", IEEE Congress on Evolutionary Computation, IEEE, 2006.

[25] Hongbing Zhu,Chengdong Pu, Eguchi, K. andJinguang Gu, "Euclidean Particle Swarm Optimization," Intelligent Networks and Intelligent Systems, IEEE, Second International Conference 1-3, 2009.

[26] Engelbrecht, Andries P. "Heterogeneous particle swarm optimization.", Springer Berlin Heidelberg, 191-202, 2010.

[27] Li-Na Qin, "Software reliability prediction model based on PSO and SVM," International Conference on Consumer Electronics, Communications and Networks (CECNet), 16-18, 2011.

[28] Malhotra, Ruchika, and Arun Negi. "Reliability modeling using Particle Swarm Optimization.", International Journal of System Assurance Engineering and Management, 2013.

[29] Jadon, Shimpi Singh, et al. "Self Adaptive Acceleration Factor in Particle Swarm Optimization.", Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications Springer India, 2013.

[30] Rita G. Al gargoor , Nada N. Saleem ,” Software Reliability Prediction Using Artificial Techniques”, IJCSI Vol. 10, Issue 4, No 2, 2013.