ISSN 2229-5518

A Survey on Software Reliability Assessment by Using Different Machine Learning Techniques

Bonthu Kotaiah , Dr. R.A. Khan

ABSTRACT: Software reliability is a special aspect of reliability engineering. System reliability, by definition, includes all parts of the system, including hardware, software, supporting infrastructure (including critical external interfaces), operators and procedures. Software reliability is a key part in software quality. The study of software reliability can be categorized into three parts: modeling, measurement and improvement. Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exist, but no single model can capture a necessary amount of the software characteristics. Assumptions and abstractions must be made to simplify the problem. There is no single model that is universal to all the situations. Software reliability measurement is naive. In this paper, we propose various machine learning approaches or techniques for the assessment of software reliability such as fuzzy approach, neuro-fuzzy approach, artificial neural network approach, genetic algorithm approach, Bayesian classification approach, support vector machine (SVM) approach, Self-organizing map approach. Also, In this paper we investigate the performance of some of the well known machine learning techniques in predicting software reliability.

Keywords: Software Reliability, Machine Learning Techniques, Assessment, Prediction, Investigation.

1. INTRODUCTION ON SOFTWARE RELIABILITY

1.1. WHAT IS SOFTWARE RELIABILITY

Software reliability is often defined as "the probability of failure-free operation of a computer program for a specified time in a specified environment." Software Reliability is also an important factor affecting system reliability. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability.

Software Reliability is an important to attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve, because the complexity of software tends to be high. While any system with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software.

1.2.HOW TO ASSESS SOFTWARE RELIABILITY

The root causes of poor reliability are found in a combination of non- compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation. Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Coding Practices
- Complexity of algorithms
- Complexity of programming practices
- Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Component or pattern re-use ratio
- Dirty programming
- Error & Exception handling (for all layers - GUI, Logic & Data)
- Multi-layer design compliance

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered

- *Bonthu Kotaiah is currently pursuing Ph.D degree program in Information Technology, Babasaheb Bhmirao Ambedkar University, Lucknow, India, PH-09666995969, 08423085961. E-mail: kotaiah_bonthuklce@yahoo.com*
- *Dr.R.A. Khan is currently working as an Associate Professor in Department of Information Technology, Babasaheb Bhimrao Ambedkar University,India . E-mail:khanraees@yahoo.com*

software. When high levels of reliability need to be assured, it will be necessary to use several sources of evidence to support reliability claims. Combining such disparate evidence to aid decision making is itself a difficult task and a topic of current research. Four areas of evidence are important in terms of benefits and limitations:

1. evidence from software components and structure;
2. evidence from static analysis of the software product;
3. evidence from testing of software under operational conditions; and
4. evidence of process quality.

Machine learning algorithms have been proven to be practical for poorly understood problem domains that have changing conditions with respect to many values and regularities. Since software problems can be formulated as learning processes and classified according to the characteristics of defect, regular machine learning algorithms are applicable to prepare a probability distribution and analyze errors (Fenton and Neil, 1999; Zhang, 2000). Decision trees, artificial neural networks, Bayesian belief network and clustering techniques such as k-nearest neighborhood are examples of most commonly used techniques for software defect prediction problems (Mitchell, 1997; Zhang, 2000; Jensen, 1996).

1.2. CLASSES OF SOFTWARE RELIABILITY ASSESSMENT

Three major classes of software reliability assessment are presented.

1. Black box reliability analysis (P. 111): Estimation of the software reliability based on failure observations from testing or operation.
2. Software metric based reliability analysis (P. 115): 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 (P. 119): Evaluation of the software system reliability from software component reliabilities and the system architecture (the way the system is composed out of the components).

2. APPROACHES TO SOFTWARE RELIABILITY

2.1. FUZZY APPROACH.

Software industry suffer many challenges in developing a high quality reliable software. Many factors affect their development such as the schedule, limited resources, uncertainty in the developing environment and inaccurate requirement specification. Software Reliability Growth Models (SRGM) were significantly used to help in solving these problems by accurately predicting the number of faults in the software during both development and testing processes. The issue of building growth models was the subject of many research work. In this approach, we explore the use of fuzzy logic to build a SRGM. The proposed fuzzy model consists of a collection of linear sub-models joined together smoothly using fuzzy membership functions to represent the fuzzy model. Results and analysis based data set developed by John Musa of Bell Telephone Laboratories are provided to show the potential advantages of using fuzzy logic in solving this problem.

2.1.1. THE SOFTWA RE RELIABILITY DATA

John Musa of Bell Telephone Laboratories compiled a software reliability database . His objective was to collect fault interval data to assist software managers in monitoring test status, predicting schedules and to assist software researchers in validating software reliability models. These models are applied in the discipline of software reliability engineering. In our case, we used data from three different projects. They are Real Time Control, Military and Operating System. A MATLAB toolbox for modeling of fuzzy systems [36] was used to implement the following results. The routines of the toolbox contain the Gustanfson-Kessel (GK) clustering algorithm.

2.1.2. VALIDATION CRITERIA

In order to check the performance of the developed model, we compute the Variance-Accounted-For (VAF) performance criterion to measure how close the measured values to the values developed using the fuzzy models. The VAF is computed as:

$$VAF = \left[1 - \frac{var(y - \hat{y})}{var(y)}\right] X\ 100\ \% \quad (1)$$

where y, $\hat{y}$ are the real actual output and the fuzzy model estimated output, respectively.

2.2. ARTIFICIAL NEURAL NETWORKS APPROACH

In this approach, we propose an artificial neural-network-based approach for software reliability estimation and modeling. We first explain the network networks from the mathematical viewpoints of software reliability modeling. That is, we will show how to apply neural network to predict software reliability by designing different elements of neural networks. Furthermore, we will

use the neural network approach to build a dynamic weighted combinational model. From experimental results, we can see that the proposed model significantly outperforms the traditional software reliability models.

2.2.1. A NEURAL-NETWORK-BASED APPROACH FOR SOFTWARE RELIABILITY MODELING

As we mentioned in the previous section, the objective function of the neural network can be considered as compound functions. In other words, if we can derive a form of compound functions from the conventional software reliability models, we can build a neural-network-based model for software reliability.

2.2.1.1. The Derivations of Proposed Approach in Software Reliability Modeling

We first consider the logistic growth curve model [3]. This model simply fits the mean value function with a form of the logistic function. Its mean value function is given by:

$$m(t) = \frac{a}{1+ke^{-bt}}, a>0, b>0, k>0. \quad (4)$$

We can derive a form of compound functions from its mean value function as the following:

Replace k with e^{-c}:

$$m(t) = \frac{a}{1+ke^{-bt}} = \frac{a}{1+e^{-c}e^{-bt}} = \frac{a}{1+e^{-(bt+c)}}.$$

Assume that

g(x)=bx+c, (5)

$$f(x) = \frac{1}{1+e^{-x}}, \quad (6)$$

and m(x)=ax. (7)

Therefore, we can get

m(f(g(x))) = m(f(bx+c))

$$= m(\frac{1}{1+e^{-(bx+c)}}) = \frac{a}{1+e^{-(bx+c)}} \quad (8)$$

This means that the mean value function of logistic growth curve model is composed of $g(x)$, $f(x)$, and $m(x)$. Subsequently, we derive the compound functions from the viewpoints of neural network. Consider the basic feed-forward network shown in Fig. 3. Note that the network has only one neuron in each layer and w_{11}^{1}, w_{11}^{0} are the weights and b_1 and b_0 are the biases. When the input, $x(t)$,at time t is fed to the input layer, we can derive the following form.

The input of the hidden layer is:

$$h\_\text{in}(t) = w^{1}_{11}t + b1 \quad (9)$$

The output of the hidden layer is:

$$h(t) = f(h\_\text{in}(t)) \quad (10)$$

where $f(x)$ is the activation function in the hidden layer.

The input of the output layer is

$$y\_in(t) = w^{0}_{11}h(t) + b0 \quad (11)$$

The output of the output layer is

$$y(t) = g(y\_in(t)), \quad (12)$$

where $g(x)$ is the activation function in the output ayer.

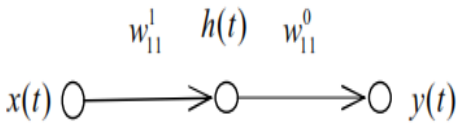

Fig. 3 Feed-forward neural net work with single neuron in each laye r

After the derivation above, we find that if we assume the activation functions $f(x)$ and $g(x)$ as:

$$f(x) = \frac{1}{1+e^{-x}}$$

$$g(x) = x$$

Furthermore, we remove the bias in the output layer. We can consequently get

$$y(t) = y\_in(t) = w^{0}_{11}h(t) \quad (13)$$

$$= w^{0}_{11} f(h\_in(t)) = \frac{w^{0}_{11}}{1+e-(w^{1}_{11}t+b1)}$$

According to Eq. (13), we have successfully derived the neural network into a logistic growth curve model. By the same process, we can derive the neural network into many other existing models.

2.3. GENETIC ALGORITHM APPROACH

Software reliability growth models (SRGMs) are very important for estimating and predicting software reliability. However, because the assumptions of traditional

parametric SRGMs (PSRMs) are usually not consistent with the real conditions, the prediction accuracy of PSRMs are hence not very satisfying in most cases. In contrast to PSRMs, the non-parametric SRGMs (NPSRMs) which use machine learning (ML) techniques, such as artificial neural networks (ANN), support vector machine (SVM) and genetic programming (GP), for reliability modeling can provide better prediction results across various projects. Gene Expression Programming (GEP) which is a new evolutionary algorithm based on Genetic algorithm (GA) and GP, has been acknowledged as a powerful ML and widely used in the field of data mining. Thus, we apply GEP into non-parametric software reliability modeling in this paper due to its unique and pretty characters, such as genetic encoding method, translation process of chromosomes. This new GEP-based modeling approach considers some important characters of reliability modeling in several main components of GEP, i.e. function set, terminal criteria, fitness function, and then obtains the final NPSRM (GEP-NPSRM) by training on failure data.

2.3.1. Software reliability modeling based on GEP

In this study, we introduce how to use GEP to extract the required non-parametric SRGM (i.e. GEP-NPSRM) from training failure data-set. There are five important com- ponents (i.e. the function set, terminal set, fitness function, control parameters and ter- mination criterion) must be determined before using GEP. Thus, the GEP-based non-parametric software reliability modeling approach is given here by considering some reliability modeling into the above five components.

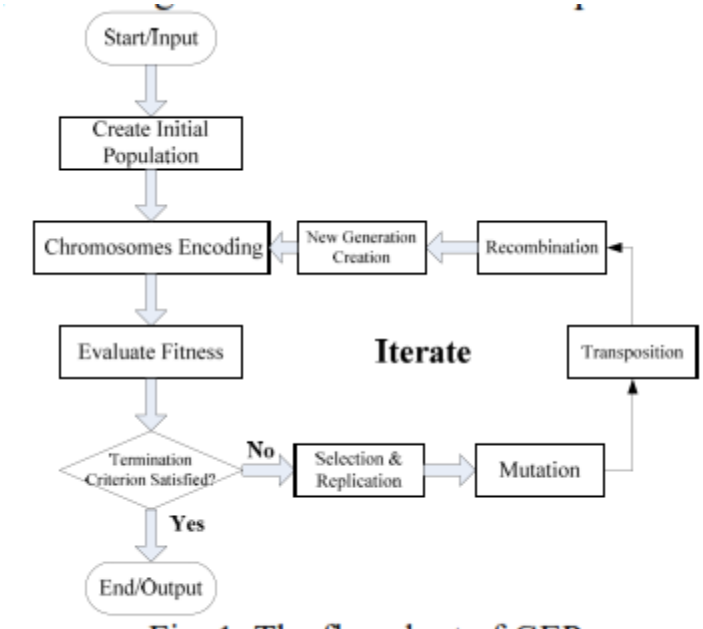

Fig. 1. The flowchart of GEP

STEPS:

Input:

1. The training failure data-set D_0 can be generally shown as two input forms $(t_1, m_1)...(t_j, m_j)...(t_n, m_n)$ or $(m_1, t_1)... (m_j, t_j)...(m_n, t_n)$, where n is data number of D_0, m_j is cumulated faults, t is failure time (interval or cumulated time). If the form of NPSRM is shown as M(t).

Data Pre-process

Because of the complexity and uncertainty of testing process, the original failure data-set unavoidably contains much noise which may affect the prediction accuracy. Thus the initial failure data-set should be pre-processed first. Besides, we also recommend several denoising methods, such as K-order moving aver- age (recommended in [4]) or exponential smoothing, for data pre-processing.

2.3.2. Modeling Process

Step1: The initial population P_0 can be created by some initialization strategy. If P_0 has the dominant characters (i.e. the genes are more diversified and suitable for the modeling object), the evolutional efficiency and the modeling quality can be effectively improved. Thus, for creating P_0 with dominant characters, we recommended several elementary functions as the elements of function set Fs, which are frequently used for software reliability modeling and shown as follows:

$$Fs=\{+,-,/,*,\exp(x),Sqrt,Log\}$$

For further validating that the function set Fs shown in Eq.2 is indeed more suitable for non-parametric reliability modeling, in section 4.1, we also compare the Fs with the function set Fs′ shown in Eq.3 which is composed of several general and elementary functions. These primary functions are also commonly used in mathematic modeling. Thus we select Fs′ as an additional function set in this paper for comparison.

$$Fs'=\{+,-,/,*,10^x,\sin,\cos\}$$

In the same way, because the GEP-NPSRM is used for reliability prediction, we recommend that the terminal set is compound by the failure time or the number of cu- mulated faults [4] in the training data-set and the random constant between 0 and 9.

Step2: Encoding chromosomes.

Step3: Fitness evaluation. The form of fitness function heavily depends on the type of problem

and must take into account that GEP was developed to maximize the fitness. Thus, we recommend the following two fitness functions which are usually used as the comparison criteria of fitting or prediction power of SRGMs.

1.Mean Squared Error (MSE):

$$\text{MSE}= \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \qquad (4)$$

2. R-Square(R):

$$\text{R}= 1-\sum_{i=1}^{n}(y_i' - y_i)^2 / \sum_{i=1}^{n}(y_i - y_{ave})^2 \qquad (5)$$

Where y_i is the observed data, y_i' is the fitting data, y_{ave} is the average value of y_i. The value of MSE is smaller or R-Square is closer to 1, the fitness of chromosome is better.

Step4: If the fitness of chromosome doesn't satisfy the terminal criterion T, turning to Step 5. Otherwise stopping iteration and turning to the Output. We recommend the following three forms of the terminal criterion T: 1) the fitness of chromosome achieves the required value; 2) the evolution process achieves a required number of generations; 3) the value of fitness has no change during the give number of generations.

Step5: Creating new generation by selection and a series of genetic operators.

Step6: Turning to Step 2 for a new iterative process.

Output: The required GEP-NPSRM satisfying the terminal criterion T.

2.4 BAYESIAN CLASSIFICATION APPROACH

In this approach, we develop a unified approach to type I and type II software reliability models in the presence of metrics information based on the use of Bayesian nonparametric regression via neural networks. Neural network based approaches are not new in software reliability prediction, but, in general such networks have been developed as direct predictors of failures or inter failure times. However, it has been suggested that such models are often prone to overfitting and are not very good at out of sample prediction of reliability. See [2]. In contrast, our approach is based on using a simple parametric model for failure times or numbers of failures where the failure rate is modeled nonparametrically.

Consider type I software reliability models where the times between successive software failures, say T_1, T_2 ,…are observed and where it is presumed that the software is corrected, possibly imperfectly, after each failure. Then, it is natural to assume a nonhomogeneous Poisson process for failures so that we model.

$$T_i \mid \lambda_i \sim EX(\lambda_i) \qquad (1)$$

for i=1,2,… Many standard software reliability models assume this basic exponential form. For instance, the Jelinski Moranda model sets

$$\lambda_i = (N-i+1)\mu \qquad (2)$$

where N represents the number of faults in the original code, γ is the fault discovery rate and perfect fault correction is assumed.

Here, we suppose that after each software failure is observed, the code is modified and software metrics reflecting the state of the code are evaluated. Then, we relate the failure rate of the software to the software metrics as follows:

$$\log \lambda_i = g(X_i) \qquad (3)$$

Where $xi = (x_{i1},....x_{ip})^T$ are the covariates available after i-1 failures have been observed. One possibility would now be to consider a linear model for the function g (x) which implies a standard exponential regression model for the inter failure times. However, in many cases, the relation between the log failure rate and the metrics may be highly non linear and therefore, a nonparametric model should be preferred. Here we use a feed forward neural network, that is:

$$g(x) = \beta_o + \sum_{j=1}^{k}\beta_j\gamma(\gamma_j{}^T x) \qquad (4)$$ where

$$\gamma(c) = (1+\exp(-c))^{-1} \qquad (5)$$

and $\gamma_j = (\gamma_j 1,....\gamma_{jp})^T$

2.5 SUPPORT VECTOR MACHINE(SVM) APPROACH

2.5.1. INTRODUCTION

Support Vector Machines (SVM) are used to detect and exploit complex patterns in data by clustering, classifying and ranking the data. They are learning machines that are used to perform binary classifications and regression estimations. They commonly use kernel based methods to apply linear classification techniques to non-linear classification problems. There are a number of types of SVM such as linear, polynomial, sigmoid etc. In recent

years, support vector machine (SVM) [4] is a new technique for solving pattern classification and universal approximation, it has been demonstrated to be very valuable for several real-world applications [5, 6]. SVM is known to generalize well in most cases and adapts at modeling nonlinear functional relationships which are difficult to model with other techniques. Consequently, we propose to apply support vector regression (SVR) to build SRGM and investigate the conditions which are typically encountered in software reliability engineering. We believe that all these characteristics are appropriate to SRGM. SVM was introduced by Vapnik in the late 1960s on the foundation of statistical learning theory [7]. It has originally been used for classification purposes but its principle can be extended easily to the task of regression by introducing an alternative loss function. The basic idea of SVR is to map the input data x into a higher dimensional feature space F via a nonlinear mapping φ and then a linear regression problem is obtained and solved in this feature space.

2.5.2. Modeling the Software Reliability Growth

In this section, we present real projects to which we apply SVR for software re- liability growth generalization and prediction. The data sets are Sys1 and Sys3 software failure data applied for software reliability growth modeling in [2]. Sys1 data set contains 54 data pairs and Sys3 data set contains 278 data pairs. The data set are normalized to the range of [0,1] first. The normalized successive failure occurrence times is the input of SVR function and the normalized accu- mulated failure number is the output of SVR function. We denote the SVR-based software reliability growth model as SVRSRG.

Here we list the math expression of three conventional SRGMs refered in the experiments.

- Goel-Okumoto Model:

$$m(t) = a(1 - e^{rt}), \quad a > 0, \quad r > 0 \tag{9}$$

- Yamada Delayed S-Shaped Model:

$$m(t) = a(1 - (1 + rt)e^{-rt}) \tag{10}$$

The approach taken to perform the modeling and prediction includes following steps:

1. Modeling the reliability growth based on the raw failure data
2. Estimating the model parameters
3. Reliability prediction based on the established model

Three groups of experiments have been performed. Training error and test- ing error have been used as evaluation criteria. In the tables presented in this paper, the l training error and the testing error are measured by sum-of-square $\sum_{i=1}^{l}(x_i - \hat{x}_i)$, where x_i, $\hat{x}_i$ are, respectively, the data set measurements and their prediction. In default case, SVR used in the experiment is ν-SVR and the parameters ν and C are optimized by cross-validation method.

In the experiment of generalization, we partition the data into two parts:

Training set and test set. Two thirds of the samples are randomly drawn from the original data set as training set and remaining one third of the samples as the testing set. This kind of training is called generalization training [8].

3 CONCLUSION

We have shown that the above discussed approaches can be used to assess the Software Reliability effectively and efficiently. The machine learning techniques can be used for building software reliability growth models. The entire system of software reliability research is considered useful for software development and testing industry. At the present we are investigating the use of different other machine learning techniques like decision-region approach, self-organization map approach, neuro-fuzzy approaches to solve the software reliability growth modeling problems. As a future work, different machine learning algorithms or improved versions of the used machine learning algorithms may be included in the experiments.

4.REFERENCES:

1. [Musa97] John D. Musa, Introduction to Software Reliability Engineering and Testing, 8th International Symposium on Software Reliability Engineering (Case Studies). November 2-5, 1997. Albuquerque, New Mexico.
2. Fengzhong Zou, Joseph Davis- Knowledge Management Research Group, School of Information Technologies, the university of Sydney, Australia.
3. [Feuring94] Feuring, Th., Lippe, W.-M., Fuzzy Neural Networks Are Universal Approximators, submitted to: IFSA World Congr. July 1995, Sao Paulo
4. [Feuring95] Feuring, Th., Fuzzy-Neuronale Netze, Ph.D. Thesis, Westf. Wilhelms Universit¨at M¨unster, Institut f¨ur Numerische u. instrumentelle Mathematik, Germany, 1995
5. [Hornik89] Hornik, K., Stinchcombe, M., Multilayer feedforward networks are universal approximators, Neural Networks, 2, pp. 359-366, 1989
6. [Lippe94] Lippe, W.-M., Feuring, Th., Tenhagen, A., A Fuzzy-Controlled Delta Bar-Delta Learningrule,

Proc. of the IEEE, Intern. Conf. on Neural Networks, pp. 1686-1690, Orlando, Fl., 1994.

7. A. L. Goel and K. Okumoto, "Time-dependent error- detection rate model for software reliability and other performancemeasures," IEEE Transactions on Reliability, Vol. 28, No. 3, pp 206–211, 1979.
8. H. Ohetera and S Yamada., "Optimal allocation and control problems for software testing resources", IEEE Transactions on Reliability, Vol. 39, No. 2, p. 171-176,1990.
9. H. Pham, System Software Reliability, Reliability Engineering Series, Springer, 2006.
10. J. G. Shanthikumar, , "A General Software Reliability Model For Performance Prediction, Microelectronics Reliability", Vol. 21, pp. 671–682, 1981.
11. Bai, C. G., Hu, Q. P., Xie, M. and Ng, S. H. (2005). Software failure prediction based on a Markov Bayesian network model. The Journal of Systems and Software, 74,275-282.
12. Cai, K. Y., Cai, L., Wang, W. D., Yu, Z. Y. and Zhang, D. (2001). On the neural network approach in software reliability modeling, The Journal of Systems and Software,58, 47-62.
13. Catal, C. and Diri, B. (2009). A systematic review of software fault prediction studies.Expert Systems with Applications, 36, 7346-7354.
14. Celeux, G., Forbes, F., Robert, C. P. and Titterington, D. M. (2007). Deviance information criteria for missing data models (with discussion). Bayesian Analysis, 1,651-706.
15. Fenton, N. and Neil, M. (1999). Software metrics: Successes, failures and new directions. Journal of Systems and Software, 47, 149-157.
16. M. R. Lyu, Handbook of Soft ware Reliability Engineering. McGraw-Hill, 1996.
17. J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability, Measurement, Prediction and Application. McGraw-Hill, 1987.
18. M. Xie, Software Reliability Modeling. World Scientific Publishing,1991.
19. C. Y. Huang, M. R. Lyu, and S. Y. Kuo, "A Unified Scheme of Some Nonhomogenous Poi sson Process Models for Soft ware Reliability Estimation," IEEE Trans. Software Eng., Vol. 29, No. 3, March 2003, pp. 261-269.
20. Z. Jelinski, and P. B. Moranda, "Software Reliability Research," in Statistical Computer Performance Evaluation (ed. Freiberger, W.), Academic Press, New York, 1972, pp. 465-484.
21. J. Musa, "Data analysis center for software: An information analysis center," Western Michigan University Library, Kalamazoo, Michigan,1980.
22. A. Sheta, "Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects," Journal of Computer Science, USA, vol. 2, no. 2, pp. 118–123, 2006.
23. T. Minohara and Y. Tohma, "Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms," in Proceedings of the 6th International Symposium on Software Reliability Engineering, pp. 324–329, 1995.
24. S. Aljahdali, D. Rine, and A. Sheta, "Prediction of software reliability: A comparison between regression and neural network non-parametric models," in ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2001), Beirut, Lebanon, pp. 470–473, 2001.
25. S. Aljahdali, A. Sheta, and D. Rine, "Predicting accumulated faults in software testing process using radial basis function network models," in 17th International Conference on Computers and Their Applica- tions (CATA), Special Session on Intelligent Software Reliability, San Francisco, California, USA, 2002.