Evolution of Software Reliability Growth Models: A Comparison of Auto-Regression and Genetic Programming Models

Mohammed Alweshah
Al-Balqa Applied University
Salt, Jordan

Walid Ahmed
Arab Academy for Science
Tech. & Maritime Transport
Alexandria, Egypt

Hamza Aldabbas
Al-Balqa Applied University
Salt, Jordan

ABSTRACT

Building reliability growth models to predict software reliability and identify and remove errors is both a necessity and a challenge for software testing engineers and project managers. Being able to predict the number of faults in software helps significantly in determining the software release date and in effectively managing project resources. Most of the growth models consider two or three parameters to estimate the accumulated faults in the testing process. Interest in using evolutionary computation to solve prediction and modeling problems has grown in recent years. In this paper, we explore the use of genetic programming (GP) as a tool to help in building growth models that can accurately predict the number of faults in software early on in the testing process. The proposed GP model is based on a recursive relation derived from the history of measured faults. The developed model is tested on real-time control, military, and operating system applications. The dataset was developed by John Musa of Bell Telephone Laboratories. The results of a comparison of the GP model with the traditional and simpler auto-regression model are presented.

Keywords

Software reliability, genetic programming, evolutionary Computation

1. INTRODUCTION

For many software companies, building software that can cope with various changes and different working environments is vital to software development and maintenance. This is why huge investment is needed to carefully test software and thus provide fault-free software. To illustrate how failures in software can have serious effects on people's lives, we give a few examples here. First, in 1985 and 1986, the Therac-25 radiation therapy machine suffered from a failure in machine design and a software failure in its control systems. This serious situation affected several patients' lives. Second, in the UK in 1992, the computer-aided dispatch system of London Ambulance Service broke down immediately after its installation. This is one of the largest ambulance services in the world and also put lives at risk. Third, on 4 June 1996, the Ariane 5 launcher failed on its maiden flight due to a software failure that occurred when an attempt to convert a 64-bit floating point number to a signed 16-bit integer caused the number to overflow. Unfortunately, the backup software was a copy and behaved in exactly the same way. Thus total system failure was a direct result of software failure [1]. The primary reason for the failure of Ariane 5 was its improper reuse of a subsystem from Ariane 4 that was adopted in order to reduce development and deployment costs. Thus, adequate testing of software to avoid such failures is an absolute necessity.

Many software techniques have been developed to assist in testing software before its release for public use. Most of these techniques are simply based on building prediction models that have the ability to predict future faults under different testing conditions [2-7]. These models are normally called software reliability growth models or SRGMs. The issue of building growth models has been the subject of many articles [8-12]. Serious applications for SRGMs such as testing software for weapon systems and NASA space shuttle applications have also been explored [13-15].

In 2001, [16], Aljahdali, et al. presented the initial idea of building an artificial neural network (ANN) model to predict software reliability. The proposed ANN model considered the historical measurements of faults as input and the predicted faults as output. The developed model successfully predicted the expected faults based on the history of four previous faults. The dataset used in the experiments was that developed by John Musa of Bell Telephone Laboratories (which can be accessed at http://johnmusa.com/ARTweb.htm. We also use this dataset to test our proposed model on real-time control, military, and operating system applications later in the paper.

Aljahdali et al. (2001) have made contributions to software reliability growth prediction using neural networks and have obtained better results compared to existing approaches with respect to predictive performance. The author have also made contributions to software reliability prediction using neural networks and have gained better results compared to the traditional analytical models with respect to predictive performance [17].
Research work on evolutionary computation (EC) and software reliability expanded in the following years. For instance, the particle swarm optimization (PSO) algorithm has been used to estimate the parameters of SRGMs and was found to have significant advantages in handling a variety of modeling problems such as the exponential model (EXPM), power model (POWM) and Delayed S-Shaped model (DSSM) [18].

The use of fuzzy logic to build a SRGM has also been explored in [6], who proposed a fuzzy model which consists of a collection of linear sub-models joined together smoothly using fuzzy membership functions to represent the fuzzy model. A comparison of various SRGMs, which included Yamada S-shaped, generalized Poisson, non-homogeneous Poisson process (NHPP), and Schneidewind reliability models was presented in [19].

The most well-known SRGMs are the logarithmic, exponential, power, S-shaped and inverse polynomial models [20-24]. However, one of the most famous models, which has been extremely well studied, is the hyper geometric distribution model. This model was proposed to estimate the number of faults/failures initially resident in a program at the start of the testing process. This model structure among other known models has three tuning parameters [25].

The fault prediction process in SRGMs depends on representing the relationship between execution time (or calendar time) and the failure count [19]. A number of unknown parameters such as the expected number of failures and the failure intensity are estimated using either the least-square or maximum likelihood estimation techniques. The model parameters are normally in nonlinear relationships. This means that traditional parameter estimation techniques suffer from many problems in finding the best set of parameters to tune the model for better prediction.

Therefore in this paper, we explore the use of genetic programming (GP) to predict the faults observed during the software testing process based on the historical data of software faults. The basic idea is to predict the next expected fault by using arbitrary measurements of previous measured faults. This is the basic idea in all regression models. Genetic programming is used here to build a suitable dynamic model structure that can better predict the faults in software.

The remainder of the paper is structured as follows: first we provide some background on EC and GP and explore the advantages of using GP in solving this problem. Second provides details of the software reliability dataset used in this paper. Finally we evaluation the developed results and conclusion.

2. EVOLUTIONARY COMPUTATION

EC is a class of adaptive stochastic search algorithms, which includes genetic algorithms (GAs) [26], evolutionary programming (EP) [27], evolution strategies (ESs) [28] and genetic programming (GP) [29]. Evolutionary computation algorithms normally operate on a population of candidate solutions. They search the space of all possible solutions till the optimal one is reached. Evolutionary computation techniques have been used successfully to solve various software reliability problems. Genetic algorithms have been used to estimate the parameters of the hyper geometric distribution model [30]. Genetic programming has also been explored in relation to its potential to solve software testing and reliability problems. [31] present some results related to software testing based on GP for medical applications. In their work, they present an automation process, where the method counts on building as intelligent systems that can be used to identify potentially dangerous software modules. However, [32] claim that predicting the exact number of faults in software is difficult and unnecessary. This viewpoint motivated them to apply GP to build a software quality classification model based on the metrics of software modules. Their GP-based model was used to distinguish fault-prone modules from non-fault-prone modules. They demonstrated their proposed model by using two case studies and showed that that GP technique can achieve good results. They also compared GP modeling and regression modeling to verify the usefulness of GP.

2.1 Genetic Programming

GP is an EC technique in which a population of candidate solutions evolves toward having a structural expression that minimizes or maximizes some evaluation function. Genetic programming has been applied to solve a variety of optimization problems including the identification of nonlinear systems [33]; [34] and in the identification of chemical processes [35]. It has also been used in the development of signal processing algorithms [36].

Genetic programming starts with a population in a tree structure, each node of which represents a computer program. Each node in the tree has a number of children which each represent a defined operation, such as plus or minus. The children of a node are evaluated and the results are directed to the next higher level of the tree. The advantage of GP is that the evolved model structure can be represented graphically in a tree structure format or in LISP format. This structure represents the best possible solution the GP can evolve. Normally, there exists a population of these solutions (i.e. tree structures) that evolves by using crossover and mutation operators [37]. Figure 1 below provides a flow chart that describes the GP technique as presented in [38].

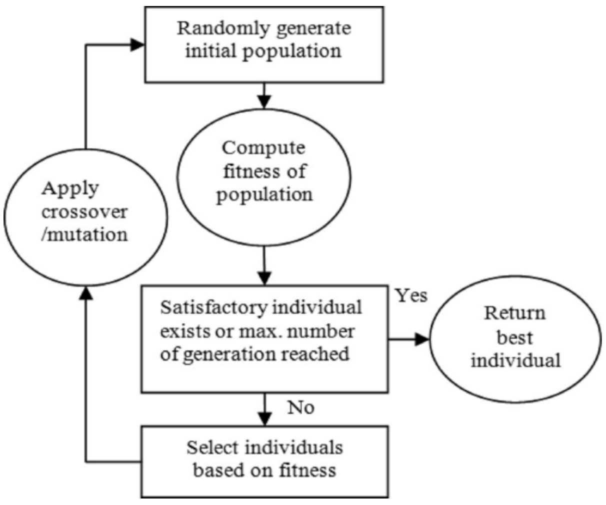
Randomly generate initial population
Compute fitness of population
Apply crossover /mutation
Satisfactory individual exists or max. number of generation reached
Yes
Return best individual
No
Select individuals based on fitness

Fig. 1. Flow chart of GP technique [38].

Genetic programming can be used to produce mathematical expressions from a database of nonlinear mathematical functions [39]. It allows the optimization of a tree structure by using a set of tuning parameters (i.e. crossover and mutation) to develop a mathematical expression. This tree structure has the advantages of having variable depth and a number of nodes. There are two types of nodes: (i) terminal nodes, which represent the input variables or a constant and (ii) function nodes, which perform some operation on the terminal nodes.

Crossover is the process where branches from two parent structures are swapped simultaneously. Mutation is the process of mutating a tree terminal or an internal swap of the parents. For each generation, the population of solutions (i.e. trees) undergoes crossover, mutation and selection. The trees are evaluated based on their fitness. The fittest solutions are kept for the next generation while the others are allowed to die. Thus, the next generation is iteratively created, and the process continues until the best structure is reached.

3. SOFTWARE RELIABILITY DATA

The software reliability dataset used in this paper was compiled by John Musa of Bell Telephone Laboratories [40]. J. Musa collected failure interval data to help software project managers in monitoring the test process, predicting project duration. The dataset contains software failure data for 16 projects. They include projects in real-time command and control, real-time commercial, military, operating systems, time sharing systems and word processing.

The dataset is available at the Data Analysis Center for Software (DACS) web page [41]. The data was collected throughout the middle 1970s. It represents projects from a variety of applications including real time command and control, word processing, commercial, and military applications. In our case, we used data from three different projects given in Table 1.

Table 1: Software Reliability Data

Application Type	Size	No. of Failures
Real-Time Command & Control	21,700	136
Military Applications	180,000	101
Operating System	Hundreds of Thousands	277

Auto-Regression Model: AR model can be presented as follows:

$$y_k = \sum_{i=1}^{n} a_i y(k-i) \ + a_0$$

where y(k–i) is the previous observed number of faults and (i =1, 2, .., n). The value of n is referred to as the “order” of the model. a0 and ai, (i =1, 2, .., n) are the model parameters.

In the following sections, for the sake of simplicity, we will substitute y(k−1), y(k− 2), y(k−3) and y(k−4) by x1, x2, x3 and x4 respectively. We used least square estimation to estimate various model parameters. We will introduce various model structures developed based regression model.

An AR model of order four was developed to predict the expected number of faults for the test/debug data of a program for the three software application of concern. The model structure and estimated parameters using least-square estimation is given in Table 2.

Table 2: Auto-Regression Models

Software Application	AR Model
Real-Time Control Applications	y(k)=0.8898x1 +0.0730x2 − 0.1549x3 +0.1612x4+2.3977
Military Applications	y(k) = 1.0087x1 − 0.0181x2 − 0.2301x3+ 0.2249x4 + 3.7427
Operating Systems Applications	y(k) = 1.0621x1 − 0.0841x2 + 0.2673x3− 0.2392x4 + 0.4034

3.1 Genetic Programming Models

We run GP with the tuning parameters presented in Table 3. They include population size, maximum number of generation, crossover and mutation parameters. The evolved model structures were presented in LISP expression based on the GP tool presented in [38].

We developed GP model structures to predict the values of the expected number of faults for the real time control, military and operating systems applications. The model structures are given by the following equations in LISP expression.

Table 3: GP Tuning Parameters

Tuning Parameters	Values
Set of Nodes	x1, x2, x3, x4
Population Size	100
Max. no. of Generations	50
No. of Training Samples	70%
No. of Testing Samples	100%
Crossover Percentage	30%
Mutation Percentage	60%
Reproduction Percentage	10%

A. Real-Time Control Applications

```
y(k) = (+x1(/(+(+(/(+x1(/(+x1( *(+(/( *x4(+x1 59))
(+( *x1x2)( * − 18 x4)))( *12 x1))(+x2x4)))
x2))x1)(/( *x141)(+( *x1x2)( * − 18 62))))
(/( *x1x1)(+( *x1x2)( *(+ − 16 14) 59))))x1))
```

B. Military Applications

```
y(k) = (+(+(+(+(+(+(+(/x1(Cos 19))
(Exp(−(+( *x1(Sqrt(Sin x2)))
(Sin(Sqrt x2)))( *x1(Cos 19)))))
(Sinx1))(Cos 19))(Sin(Sqrt x2)))
(Sinx3))(Cos(+(+(/x1(Cos 19))(Cos x3))
(Ln(/x1(Cos x3))))))(Cos(+(+(/x1(Cos x3))
(Lnx1))(Lnx1))))
```

C. Operating Systems Applications

```
y(k) = (+(Ln(Sin(Sin(Ln(Sqrt(+(+(Sin(Sin x3))
x2)x2))))))(+(Ln(Sin(+(+(Sin(Cos(+(+(Cos
(Sqrt x1))(+(Sin(Sin x1))x2))x2)))
```

4. MODEL EVALUATION

In order to check the performance of the developed AR and GP models, the variance-accounted-for (VAF) performance criterion is used. The VAF measures the closeness between two curves or two arrays. The VAF is computed as:

VAF = [1- var(actual-estimated)/var(actual)]x100%

var represents the variance of the signal or array defined.

4.1 Evaluation of the Developed Results

Time series forecasting depends on using past measurements to build some model structure that can provide an estimates of future measurements using prediction. Principally, this approach attempts to model a nonlinear function based a recurrence relation derived from past measurements. The recurrence relation is then used to provide an approximate new measurement of the time series, which expectantly will be a good approximation of the actual measurements.

It can be seen that the developed models based the recursive concept outperform traditional models know in the literature. The computed performance criterion, the VAF is quite high. The actual and estimated responses are very close in figures. Regarding GP, there was little improvement over the recursive models. This means that GP can be used as an alternative technique to the recursive mode

Table 4: Computed VAF for both Regression and GP Models

Model	Military Application	Real-Time Application	Operating Systems Application
AR Model	99.5%	99.5%	99.96%
GP Model	99.7%	99.8%	99.97%

Figures 1, 2 and 3 show the actual measurement software faults versus the estimated regression model and the GP model faults for the three software application of concern. The computed VAF in each case is presented in Table 4.

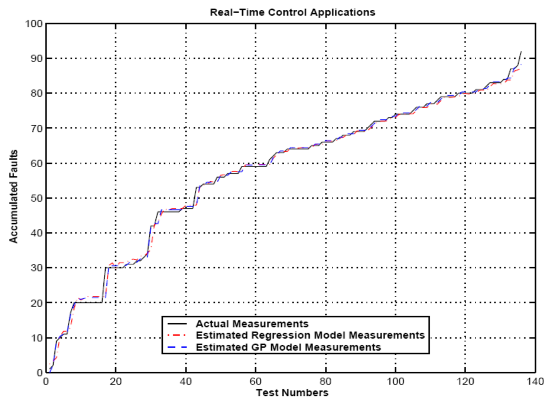

Fig. 1. Actual and estimated responses in Real-Time Control Applications

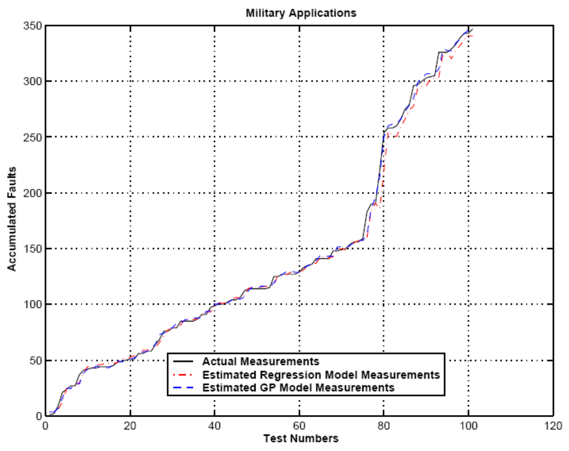

Fig. 2. Actual and estimated responses in Military Applications

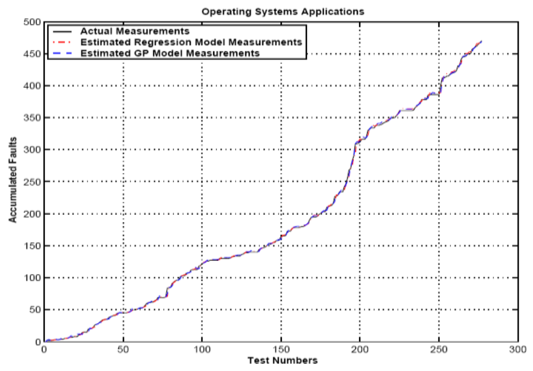

Fig. 3. Actual and estimated responses in Operating Systems Applications

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

5. A CONCLUSION AND FUTURE WORK

In this paper we explored the development of SRGM using two techniques; the traditional Auto-Regression simple linear model and genetic programming evolutionary models. The models were used to predict the expected number of faults at the current instance of time based on the previous measured faults. This type of software research helps in predicting the software reliability in various applications and helps software manager to plan its resources and the release day of the product. The developed models were tested on real measured dataset collected from John Musa of Bell Telephone laboratories. Three applications of software produce in real-time control, military and operating systems applications. We plan to expand our research to enhance the developed mathematical model complexity and explore alternative soft computing techniques.

6. REFERENCES

[1] F. H. Allen, *et al.*, "The Cambridge Crystallographic Data Centre: computer-based search, retrieval, analysis and display of information," *Acta Crystallographica Section B: Structural Crystallography and Crystal Chemistry*, vol. 35, pp. 2331-2339, 1979.

[2] K. Okumoto and A. L. Goel, "Optimum release time for software systems based on reliability and cost criteria," *Journal of Systems and Software,* vol. 1, pp. 315-318, 1980.

[3] S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements," *European Journal of Operational Research,* vol. 31, pp. 46-51, 1987.

[4] S.-Y. Kuo, *et al.*, "Framework for modeling software reliability, using various testing-efforts and fault-detection rates," *Reliability, IEEE Transactions on,* vol. 50, pp. 310-320, 2001.

[5] S. Yamada, *Software reliability modeling: fundamentals and applications*: Springer, 2014.

[6] S. Aljahdali and A. F. Sheta, "Predicting the Reliability of Software Systems Using Fuzzy Logic," in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, 2011, pp. 36-40.

[7] M. Alweshah, "Firefly Algorithm with Artificial Neural Network for Time Series Problems," *Research Journal of Applied Sciences, Engineering and Technology,* vol. 7, pp. 3978-3982, 2014.

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

[9] S. Brocklehurst*, et al.*, "Recalibrating software reliability models," *Software Engineering, IEEE Transactions on,* vol. 16, pp. 458-470, 1990.

[10] M. R. Lyu, *Handbook of software reliability engineering* vol. 222: IEEE computer society press CA, 1996.

[11] A. Sheta, "Reliability growth modeling for software fault detection using particle swarm optimization," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 3071-3078.

[12] E. A. El-Sebakhy, "Software reliability identification using functional networks: A comparative study," *Expert systems with applications,* vol. 36, pp. 4013-4020, 2009.

[13] N. F. Schneidewind and T. W. Keller, "Applying reliability models to the space shuttle," *Software, IEEE,* vol. 9, pp. 28-33, 1992.

[14] P. Carnes, "Software reliability in weapon systems," in *Software Reliability Engineering-Case Studies, 1997. Proceedings., The Eighth International Symposium on*, 1997, pp. 95-100.

[15] T. Keller and N. F. Schneidewind, "Successful application of software reliability engineering for the NASA space shuttle," *Computer Standards & Interfaces,* vol. 21, pp. 169-170, 1999.

[16] S. H. Aljahdali*, et al.*, "Prediction of software reliability: A comparison between regression and neural network non-parametric models," in *Computer Systems and Applications, ACS/IEEE International Conference on. 2001*, 2001, pp. 470-473.

[17] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," *Reliability Engineering & system safety,* vol. 87, pp. 45-51, 2005.

[18] A. Sheta and J. Al-Salt, "Parameter estimation of software reliability growth models by particle swarm optimization," *management,* vol. 7, p. 14, 2007.

[19] Z. ALRahamneh*, et al.*, "A New Software Reliability Growth Model: Genetic-Programming-Based Approach," *Journal of Software Engineering and Applications,* vol. 4, p. 476, 2011.

[20] B. Littlewood and J. L. Verrall, "A Bayesian reliability model with a stochastically monotone failure rate," *Reliability, IEEE Transactions on,* vol. 23, pp. 108-114, 1974.

[21] J. D. Musa, "A theory of software reliability and its application," *Software Engineering, IEEE Transactions on,* pp. 312-327, 1975.

[22] J. D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," in *Proceedings of the 7th international conference on Software engineering*, 1984, pp. 230-238.

[23] S. Yamada*, et al.*, "S-shaped software reliability growth models and their applications," *Reliability, IEEE Transactions on,* vol. 33, pp. 289-292, 1984.

[24] N. F. Schneidewind, "Measuring and evaluating maintenance process using reliability, risk, and test metrics," *Software Engineering, IEEE Transactions on,* vol. 25, pp. 769-781, 1999.

[25] Y. Tohma*, et al.*, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," *Software Engineering, IEEE Transactions on,* vol. 15, pp. 345-355, 1989.

[26] J. Holland, "Adaptation in Natural and Artificial Systems, Ann Arbor: University of Michigan Press, 1975," 1992.

[27] L. J. Fogel*, et al.*, "Artificial intelligence through simulated evolution," 1966.

[28] I. Rechenberg, "Evolutionsstrategie-optimierung technischer systems nach prinzipien der biologischen evolution, stuttgart: Frommannholzboog, 1973," ed: New York: John Wiley, 1981.

[29] J. R. Koza, "Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm," in *ICGA*, 1991, pp. 37-44.

[30] T. Minohara and Y. Tohma, "Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms," in *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, 1995, pp. 324-329.

[31] V. Podgorelec*, et al.*, "Testing reliability of medical software," in *Computer-Based Medical Systems, 2002.(CBMS 2002). Proceedings of the 15th IEEE Symposium on*, 2002, pp. 185-190.

[32] Y. Liu and T. M. Khoshgoftaar, "Genetic programming model for software quality classification," in *High Assurance Systems Engineering, 2001. Sixth IEEE International Symposium on*, 2001, pp. 127-136.

[33] G. J. Gray*, et al.*, "Structural system identification using genetic programming and a block diagram oriented simulation tool," *Electronics Letters,* vol. 32, pp. 1422-1424, 1996.

[34] A. Sheta*, et al.*, "Development of software effort and schedule estimation models using soft computing techniques," in *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, 2008, pp. 1283-1289.

[35] K. D. Bettenhausen*, et al.*, "Self-organizing structured modelling of a biotechnological fed-batch fermentation by means of genetic programming," in *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, 1995, pp. 481-486.

[36] K. C. Sharman*, et al.*, "Evolving signal processing algorithms by genetic programming," in *Genetic Algorithms in Engineering Systems: Innovations and*

Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414), 1995, pp. 473-480.

[37] E. O. Costa, *et al.*, "A genetic programming approach for software reliability modeling," *Reliability, IEEE Transactions on,* vol. 59, pp. 222-230, 2010.

[38] H. Faris, *et al.*, "Modelling hot rolling manufacturing process using soft computing techniques," *International Journal of Computer Integrated Manufacturing,* vol. 26, pp. 762-771, 2013.

[39] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection* vol. 1: MIT press, 1992.

[40] J. Musa, "Data analysis center for software: An information analysis center," *Western Michigan University Library, Kalamazoo, Michigan*, 1980.

[41] T. McGibbon, "A business case for software process improvement revised," *DoD Data Analysis Center for Software (DACS),* 1999.