Evolutionary Regression Prediction for Software Cumulative Failure Modeling: a comparative study

M. Benaddy[1], M. Wakrim[1] & S. Aljahdali[2]
1 : Dept. of Math. & Info. Equipe MMS, Ibn Zohr University Morocco. benaddym@yahoo.fr
2: Taif University Saudi Arabia

***Abstract*— An evolutionary regression modeling approach for software cumulative failure prediction based on auto-regression order 4, 7 and 10 models are proposed. A real coded genetic algorithm is used to optimize the mean square of the error produced by training the auto-regression model. In this paper, we present a real coded genetic algorithm that uses the appropriate operators for this encoding type to train the auto-regression model. To evaluate the predictive capability of the developed model data sets, various projects were used. A comparison between auto-regression order 4 model trained using least square estimation [1] and real coded genetic algorithm training is provided, also a comparison between the auto-regression order 7 and 10 models trained using the genetic algorithm is presented. Experimental results show that the training of different auto-regression model by the real coded genetic algorithm has a good predictive capability.**

Keywords— *Genetic Algorithms, Real Coded Genetic Algorithms, Auto Regression Model, Software Reliability.*

I. INTRODUCTION

Software reliability is defined as the probability of failure free software operation for a specified period of time in a specified environment [3]. Society's reliance on large complex systems mandates high reliability. Reliable software is a necessary component. Controlling faults in software requires that one can predict problems early enough to take preventive action. In the past 35 years more than 100 software reliability models have been developed to solve reliability models [12]. Most of these models as the models of software reliability growth depend on a certain a priori assumptions about the nature of software faults and the stochastic behavior of software process [4]-[5]. As a result, different models have different predictive performance at different testing phases across various projects. A single universal model that can provide highly accurate predictions under all circumstances without any assumptions is most desirable [15]-[11]. Regression models are the most popular method for building models and are used to calibrate almost all of the models [2]. Regression prediction models are one of the proposed models to predict the number of faults in the software as shown by Aljahdali [1]-[2].

Least square estimation is the most technique used to estimate linear models as observed in the literature [1]. In this paper we have developed a real coded genetic algorithm (RCGA) as an alternative to estimate the auto-regression model proposed by Aljahdali [1]-[2], that optimizes the error made by the model. Evolving the parameters set for the auto-regression model with the inverted error as a fitness function. To show the effect of the order of the model, an auto-regression model order 7 and 10 is established by using our proposed approach to estimate its parameters.

II. SOFTWARE RELIABILITY DATA SET:

The Software Reliability Dataset was compiled by John Musa of Bell Telephone Laboratories [6]. His objective was to collect failure interval data to assist software managers in monitoring test status and predicting schedules and to assist software researchers in validating software reliability models. These models are applied in the discipline of Software Reliability Engineering. The dataset consists of software failure data on 16 projects. Careful controls were employed during data collection to ensure that the data would be of high quality. The data was collected throughout the mid 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. They are Military, Real Time Control and Operating System. The failure data were initially stored in arrays, ordered by day of occurrence so that it could be processed.

III. REGRESSION MODELS

One of the most famous regression models is the Auto-Regressive models. This model has been used in many applications. The auto-regressive model can be described as following form [1]:

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

$y(k-\tau_i)$ is the observed cumulative failure n days before the current day, a_i is the tuning parameter for the auto-

978-1-4244-3757-3/09/$25.00 ©2009 IEEE

regression model and *n* is referred to as the 'order' of the model.

IV. The Genetic Algorithm

The genetic algorithm was developed and formalized by Holland [10]. It was further developed and shown to have wide applicability by Goldberg [8]. For a good introduction and examination of GAs, see Michalewicz [13].

Although there are many possible varieties on the basic GAs, the operational of every genetic algorithm is described in the following steps:

1. Randomly create an initial population of chromosomes.
2. Compute the fitness of every member of the current population.
3. If there is a member of the current population that satisfies the problem requirements then stop. Otherwise continue to the next step.
4. Create an intermediate population by extracting members from the current population using a selection operator.
5. Generate a new population by applying the genetic operators of crossover and mutation to this intermediate population.
6. Go back to step 2.

V. Real Coded Genetic Algorithms

The most common representation in GAs is binary [9]. The chromosomes consists of a set of genes, which are generally characters belonging to an alphabet {0, 1}. Therefore, a chromosome is a vector *C* consisting of *l* genes c_i: $C=(c_1,c_2,\ldots,c_l)$, $c_i=\{0,1\}$, Where *l* is the length of the chromosome.

However in the optimization problems of parameters with variables in continuous domains, it is more natural to represent the genes directly as a real numbers since the representation of solution are very close to the natural formulation, i.e. there are no differences between the genotype and the phenotype. The use of this real-coding in numerical optimization on continuous domains appears in Michalewicz [13].

In this case, a chromosome is a vector of floating point numbers. The chromosome length is the vector length of the solution of the problem; thus, each gene represents a variable of the problem. The gene values are forced to remain in the interval established by the variables they represent, so many genetic operators are developed for them, such as, Flat crossover [16], Arithmetic crossover [14] and BLX-α crossover [7] for the crossover operators and Random mutation and non-uniform mutation for the mutation operators [14].

VI. The Real Coded Genetic algorithm TO Estimate Auto-regression parametrs for Software Reliability Prediction

As mentioned above, real coding is the most suitable coding for continuous domains. Since our goal is auto-regression model parameter's estimating which predicts the cumulative future faults in the software, it appears logical to use this coding and genetic operators associated to it. Among the advantages of using real-valued coding over binary coding is increased precision. Binary coding of real-valued numbers can suffer loss of precision depending on the number of bits used to represent one number. Moreover, in real-valued coding chromosome string become much shorter. For real-valued optimization problems, real-valued coding is simply much easier and more efficient to implement, since it is conceptually closer to the problem space. In particular, our aim is to train an auto-regression model to predict future faults in the software from the previous discovered faults.

A chromosome consists of all the parameters. One gene of a chromosome represents a single parameter value. In our case there are; *5, 8* and 11 parameters for the auto-regression order *4*, *7* and *10* respectively. The length of the chromosomes is *l= 5, l= 8 and l= 11*. The parameters of the auto-regression are placed on a chromosome as shown in figure 1.

$$\underbrace{\theta_0\theta_1\theta_2\ldots\theta_{n-1}\theta_n}$$

Figure1: The chromosomal representation of the auto-regression model.

Fitness function: the fitness function should reflect the individual's performance in the current problem. We have chosen *1/(1+mse)* as a fitness function Eq. (3), where *mse* is the mean squared error during training defined in Eq (2).

$$mse = \frac{1}{n}\sum(\beta_i - \hat{\beta}_i)^2 \qquad \textbf{(2)}$$

Where *n* is the number of training faults used during the training process. β_i And $\hat{\beta}_i$ are the actual and the predicted output respectively during the learning process.

$$fitness = \frac{1}{1+mse} \qquad \textbf{(3)}$$

Selection mechanism: The roulette wheel selection is used to create the intermediate population. For each chromosome C_i in a population *P*, the probability $p_s(C_i)$, of including a copy of this chromosome in the intermediate population P' is calculated as in Eq. (4)

$$p_s(C_i) = \frac{fitness(C_i)}{\sum_{j=1}^{P} fitness(C_j)} \qquad \textbf{(4)}$$

Where *P* is the number of individuals in the population *P*.

VII. THE REAL CODED GENETIC ALGORITHM TRAINING AND TESTING RESULTS

The initial parameters were randomly chosen in the interval [0, 1]. For each project we performed a number of simulations with a population of 200 individuals and a maximum of generation equal to *Gmax*. After the training process the Normalize Root Square Error (*NRMSE*, see Eq. 6) is computed to compare the results obtained by real coded genetic algorithm with these obtained by the least square estimation.

$$NRMSE = \frac{1}{n}\sqrt{\frac{\sum_{i=1}^{n}(\beta(i)-\hat{\beta}(i))^2}{\sum_{i=1}^{n}(\beta(i))^2}} \quad \textbf{(5)}$$

The results of *NRMSE* obtained, by the LSE in test phase are given in a table 1.

Table 1: Results for NRMSE obtained using Auto-regression model order 4 model in testing case [3].

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	71	96	194
Testing Data	101	136	277
Regression Model	3.1434	1.7086	1.0659

The results of *MSE* and *NRMSE* obtained, by the training the Auto-regression order 4 model with our real coded genetic algorithm in training and testing phases are given in table 2.

Table 2: Results for the *MSE* and *NRMSE* obtained using *AR-4* trained by RCGA in the training and testing phases.

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	71	96	194
MSE	1.7323943	2.6185567	1.7474227
NRMSE	1.6062E-4	6.0893E-4	3.1551E-5
Testing Data	101	136	277
MSE	2.6930692	2.7867646	2.1227436
NRMSE	3.7809E-5	3.1940E-4	1.3047E-5

In figure 2 to 7 we are showing the testing results and error difference for various projects using the AR-4 trained by our real coded genetic algorithm.

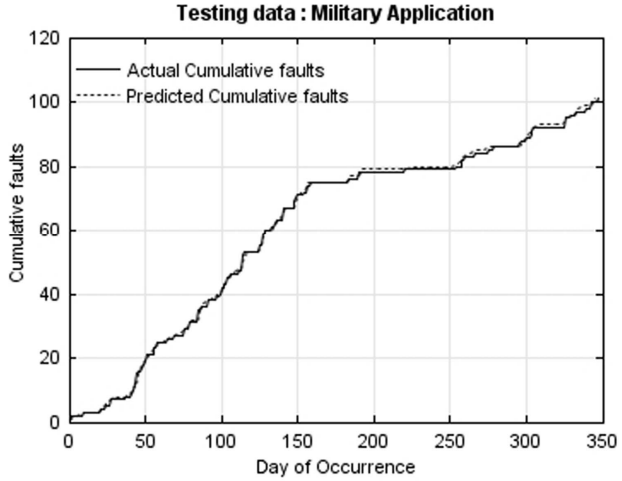

Figure 2:Actual and Predicted Faults in Testing phase: Military Application.

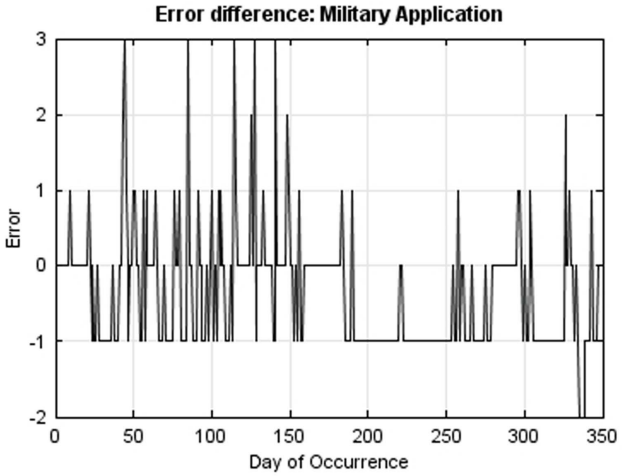

Figure 3: Prediction Error in testing phase: Military Application.

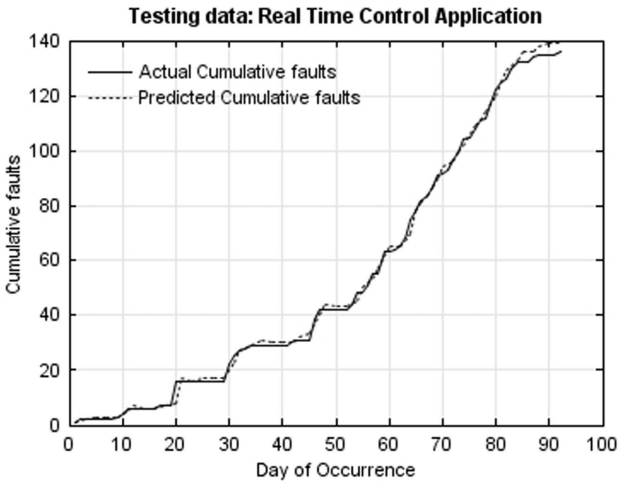

Figure 4: Actual and Predicted Faults in Testing phase: Real Time Control Application.

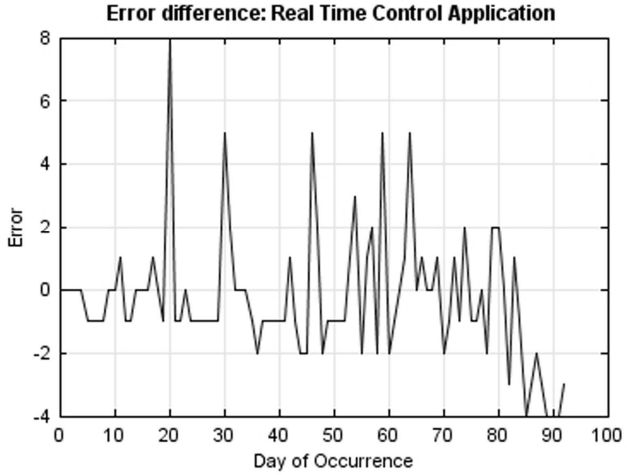

Figure 5: Prediction Error in testing phase:Real Time Control Application.

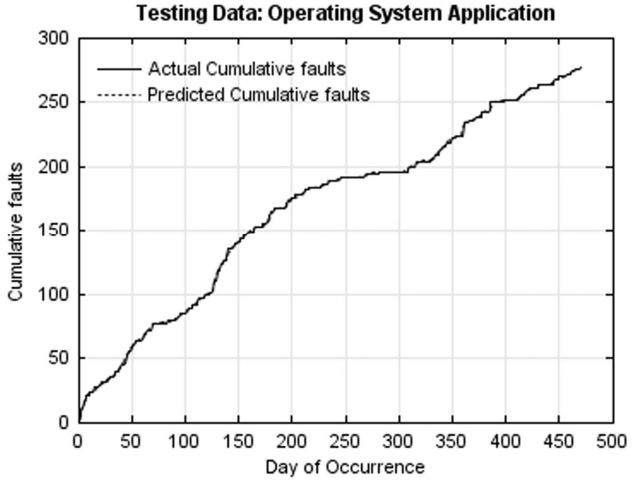

Figure 6: Actual and Predicted Faults in Testing phase: Operating System Application.

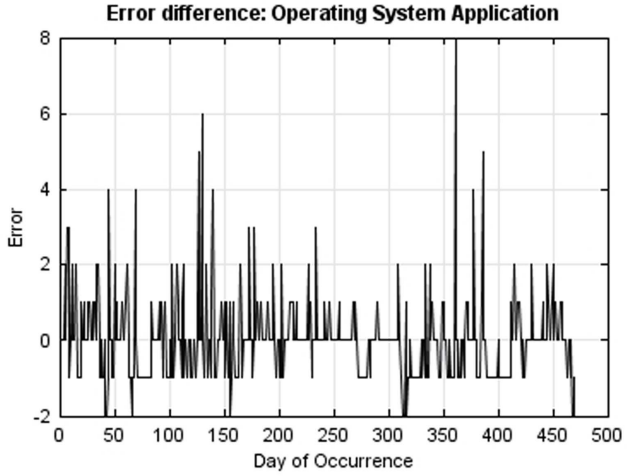

Figure 7: Prediction Error in testing phase: Operating System Application.

VIII. RESULTS FOR THE AR-7 AND AR-10 TRAINED BY THE RCGA

The results of *MSE* and *NRMSE* obtained, by the training the Auto-regression order 7 and 10 models with our real coded genetic algorithm in training and testing phases are given in tables 3 and 4.

Table 3: Results for the MSE and NRMSE obtained using AR-7 trained by RCGA in the training and testing phases.

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	71	96	194
MSE	2.6197183	3.8659794	2.3556702
NRMSE	1.9752E-4	7.3989E-4	3.6633E-5
Testing Data	101	136	277
MSE	4.4059405	8.536765	2.9530685
NRMSE	4.8361E-5	5.5903E-4	1.5388E-5

Tableau 4: Results for the MSE and NRMSE obtained using AR-10 trained by RCGA in the training and testing phases.

Project Name	Military	Real Time Control	Operating System
Number of Faults	101	136	277
Training Data	71	96	194
MSE	2.915493	1.8762887	3.9948454
NRMSE	2.0837E-4	5.1545E-4	4.7705E-5
Testing Data	101	136	277
MSE	7.643564	6.529412	5.6498194
NRMSE	6.3698E-5	4.8891E-4	2.1285E-5

In figure 8 to 13 we are showing the testing results and error difference for various projects using the AR-7 trained by our real coded genetic algorithm.

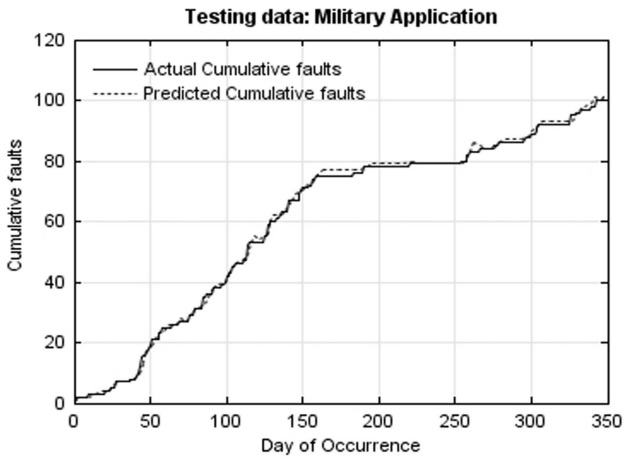

Figure 8: Actual and Predicted Faults in Testing phase: Military Application.

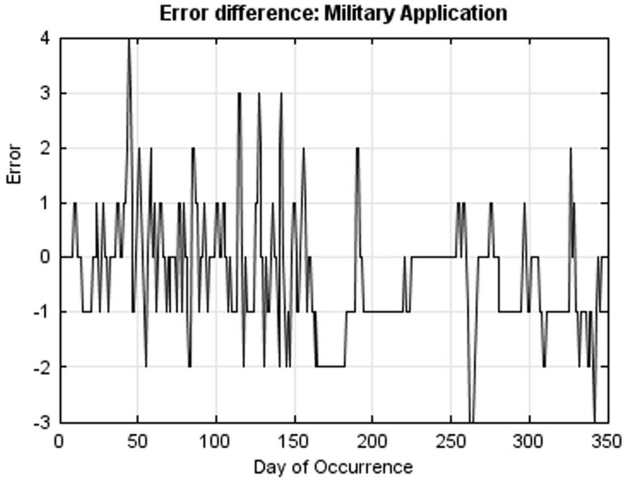

Figure 9: Prediction Error in testing phase: Military Application.

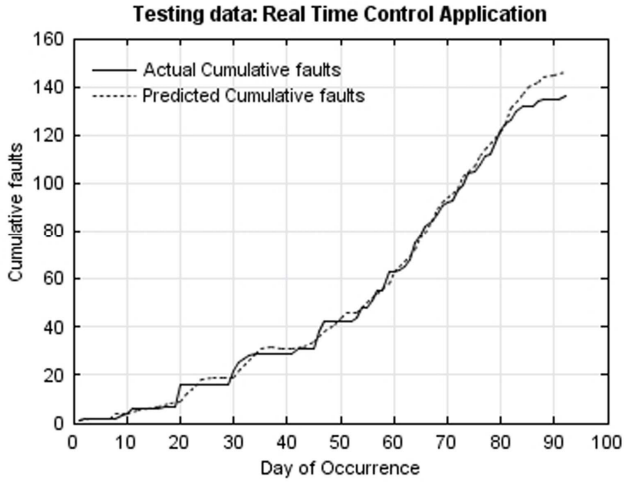

Figure 10: Actual and Predicted Faults in Testing phase: Real Time Control Application.

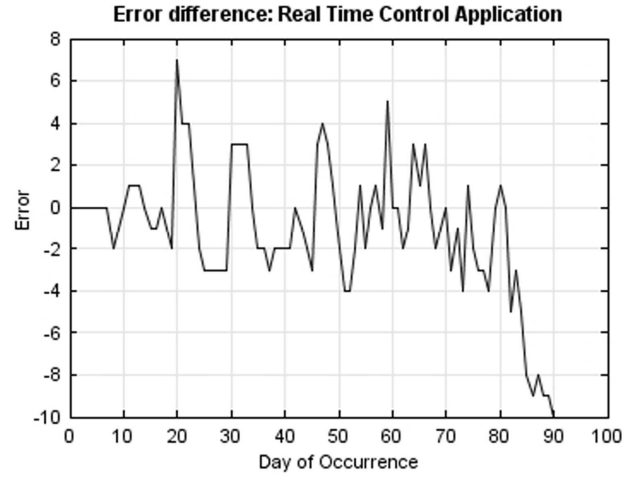

Figure 11: Prediction Error in testing phase:Real Time Control Application.

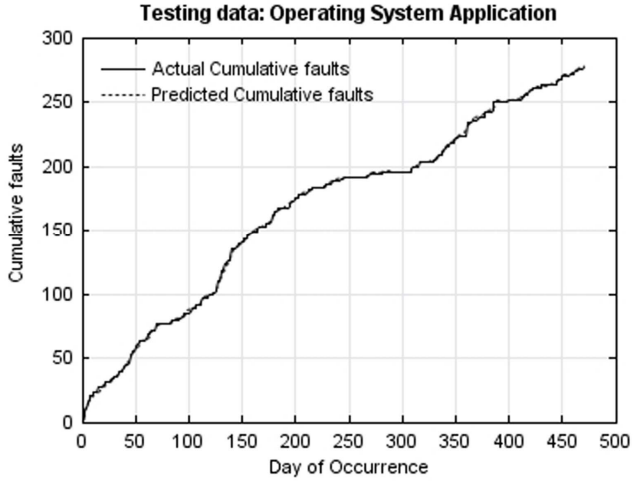

Figure 12: Actual and Predicted Faults in Testing phase: Operating System Application.

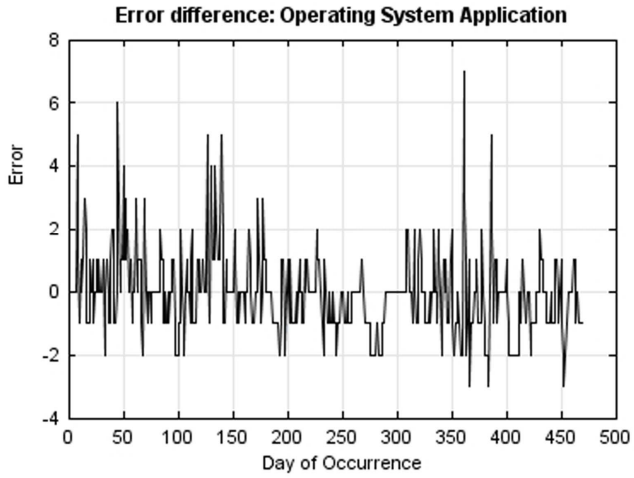

Figure 13: Prediction Error in testing phase: Operating System Application.

In figure 14 to 19 we are showing the testing results and error difference for various projects using the AR-10 trained by our real coded genetic algorithm.

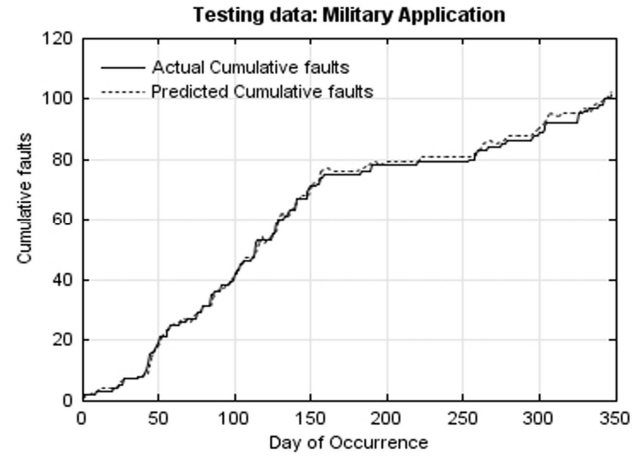

Figure 14: Actual and Predicted Faults in Testing phase: Military Application.

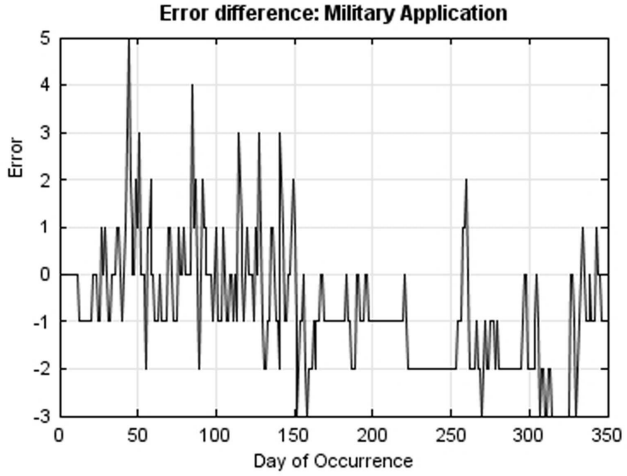

Figure 15: Prediction Error in testing phase: Military Application.

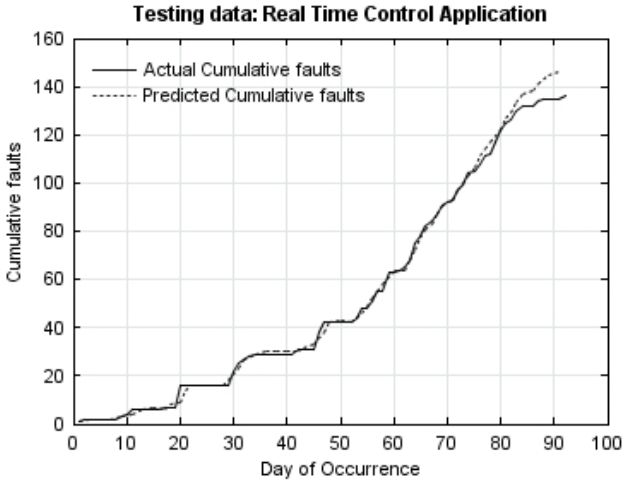

Figure 16: Actual and Predicted Faults in Testing phase: Real Time Control Application.

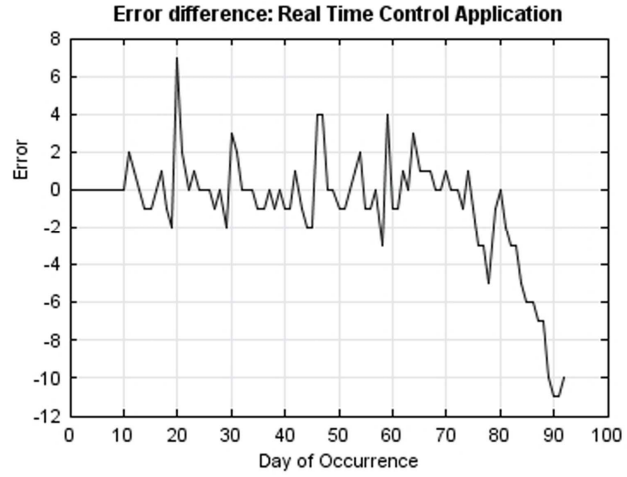

Figure 17: Prediction Error in testing phase:Real Time Control Application.

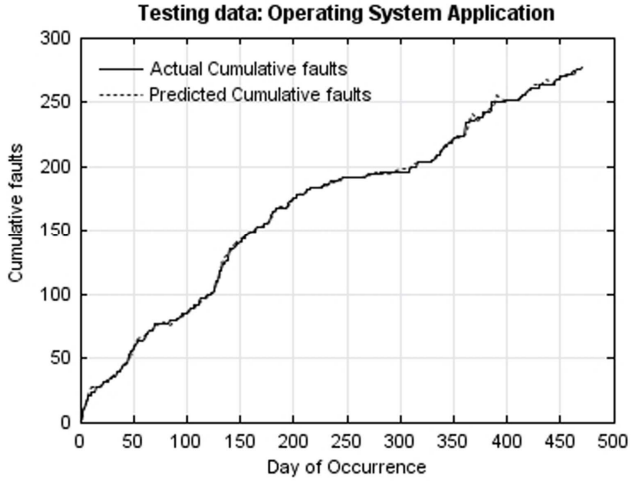

Figure 18: Actual and Predicted Faults in Testing phase: Operating System Application.

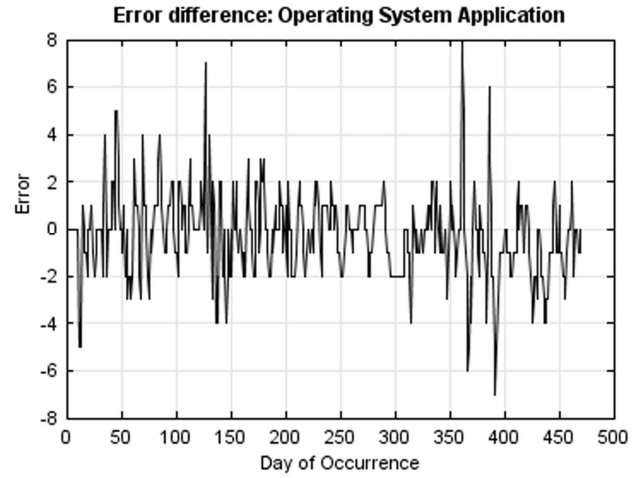

Figure 19: Prediction Error in testing phase: Operating System Application.

IX. CONCLUSION

In this paper, an evolutionary Auto-regression modeling approach for software cumulative failure prediction is proposed. Genetic algorithm is used to learn the auto-regression models by optimizing the mean square error produced by these models.

Experimental results show that our proposed approach adapts well across different projects, and has a better performance compared to the results obtained by Auto-regression models for cumulative failure, learned by the Lest Square Estimation.

The order of the auto-regression model doesn't affect the model as shown in different results for distinct projects.

REFERENCES

[1] S. Aljahdali, A. Sheta and D. Rine, "Prediction of Software Reliability: A Comparison between regression and neural

network non-parametric Models", Proceeding of the IEEE/ACS Conference, 25-29, June 2001.
[2] S. Aljahdali, K. A. Buragga, "Evolutionary Neural Network Prediction for Software Reliability Modeling" The 16th International Conference on Software Engineering and Data Engineering (SEDE-2007).
[3] ANSI /IEEE, "Standard Glossary of Software Engineering Terminology," STD-729-1991, ANSI /IEEE, 1991.
[4] K.Y. Cai, C.Y. Xen and M.L. Zhang, "A critical review on software reliability modeling", Reliability Engineering and Safety, 1991.
[5] K.Y. Cai, L. Cai, W.D. Wang, Z.Y. Yu, D. Zhang, "On the Neural Network approach in software reliability modeling," Journal of System and Software 2001.
[6] Data & Analysis Centre for Software DACS https://www.thedacs.com/databases/sled.
[7] L.J. Eshelman & J.D. Schaffer, "Real coded genetic algorithms and interval schemata" In L. Durrel Whitely, Foundation of genetic algorithms 2 (pp. 187-202). San Mateo: Morgan Kaufman.
[8] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wisley New York, 1989.
[9] D.E. Goldberg, "Real-coded genetic algorithms. Virtual alphabets and blocking." Complex Systems, 5, 1991, 139-167.
[10] J.H. Holand, "Adaptation in Natural and Artificial Systems". Cambridge, Mass: MIT press, 1975.
[11] Karunanithi N, Withtely D, Malaiya YK.. "Prediction of Software Reliability using Connectionist Models," IEEE Trans Software Eng 1992.
[12] M.R. Lyu, *"Software Reliability Engineering: A Roadmap"*. Future of Software Engineering (FOSE'07) IEEE CS Press 2007.
[13] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer 1996.
[14] Z. Michalewicz "Genetic Algorithms + Data Structures = Evolution Programs", New-York : Springer, 1992.
[15] J.Y. Park, S.U. Lee, J.H. Park, "Neural Network Modeling for Software Reliability Prediction from Failure Time Data," J Electr Eng Inform Sc 1999.
[16] N.J. Radcliffe, "Equivalence class of genetic algorithms" Complex Systems, 1991, 5 (2), 183-205.

978-1-4244-3757-3/09/$25.00 ©2009 IEEE