International Journal of Reliability, Quality and Safety Engineering
Vol. 12, No. 4 (2005) 309–321

World Scientific
www.worldscientific.com

DYNAMIC SOFTWARE RELIABILITY PREDICTION: AN APPROACH BASED ON SUPPORT VECTOR MACHINES

LIANG TIAN* and AFZEL NOORE†

Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506–6109, USA
**tian@csee.wvu.edu*
†*afzel.noore@mail.wvu.edu*

Received 23 August 2004
Revised 27 January 2005

A support vector machine (SVM) modeling approach for software reliability prediction is proposed. Based on the structural risk minimization principle, the learning scheme of SVM is focused on minimizing an upper bound of the generalization error that eventually results in better generalization performance. The SVM learning scheme is applied to the failure time data, forcing the network to learn and recognize the inherent internal temporal property of software failure sequence. Further, the SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure behavior. The performance of our proposed approach has been tested using four real-time control and flight dynamic application data sets and compared with feed-forward neural network and recurrent neural network modeling approaches. Experimental results show that our proposed approach adapts well across different software projects, and has a better next-step prediction performance.

Keywords: Support vector machines; software reliability growth prediction; failure time data.

1. Introduction

One of the best approach to evaluate and predict software reliability quantitatively is to use software reliability models. Software reliability is defined as the probability of a failure free operation of software for a specified period of time in a given environment.[1] A software reliability model is a set of mathematical equations that are used to describe the behavior of software failures with respect to time and predict software reliability performance such as the mean time between failures and the number of residual faults.[1] Most of the existing analytical software reliability growth models depend on *a priori* assumptions about the nature of software faults

†Corresponding author.

and the stochastic behavior of software failure process.[2–7] As a result, each model has a different predictive performance across various projects. A general model that can provide accurate predictions under multiple circumstances is most desirable.[4–6] It has been shown that a neural network approach is a universal approximator for any non-linear continuous function with an arbitrary accuracy.[3,8] The underlying failure process can be learned and modeled based on only failure history of a software system rather than *a priori* assumptions.[4,9] Consequently, it has become an alternative method in software reliability modeling, evaluation and prediction. Karunanithi *et al.*[4,5] were the first to propose a neural network approach for software reliability growth modeling. Adnan *et al.*,[10,11] Aljahdali *et al.*,[12,13] Ho *et al.*,[14] Park *et al.*,[6] Sitte,[15] and Tian and Noore[16,17] have also made contributions to software reliability growth prediction using neural networks, and have obtained better results compared to the existing approaches with respect to predictive performance.

Most of the published literature used neural network to model the relationship between software failure time and the sequence number of failures. Some examples are: cumulative execution time as input and the corresponding accumulated number of defects disclosed as desired output,[4,5] and failure sequence number as input and the corresponding failure time as desired output.[6] Those neural network modeling approaches adopt the gradient descent based back-propagation learning scheme to implement the empirical risk minimization (ERM) principle, which only minimizes the mean square error during the training process and thus improves the training accuracy. In this case, the focus of the training process is model fitting and tends to cause overfitting. The error on the training data set is driven to a very small value for known data, but when out-of-sample data is presented to the network, the error is unpredictably large, which yields limited generalization capability.

As a novel type of machine learning algorithm, SVM has gained increasing attention from its original application in pattern recognition to the extended application in function approximation and regression estimation.[18–20] Based on the structural risk minimization (SRM) principle, the learning scheme of SVM is focused on minimizing an upper bound of the generalization error that includes the sum of the empirical training error and a regularized confidence interval, which will eventually result in better generalization performance. Moreover, unlike other gradient descent based learning scheme with the danger of getting trapped into local minima, the regularized risk function of SVM can be minimized by solving a linearly constrained quadratic programming problem, which can always obtain a unique and global optimal solution. Thus, the possibility of being trapped at local minima can be effectively avoided.[19,21–23]

In this paper we propose a dynamic software reliability prediction model using support vector machines. We model the inter-relationship among software failure time. The SVM learning scheme is applied to the failure time data, forcing the network to learn the inherent internal temporal property of software failure sequence. The SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure sequence. Further, the generalization

capability of the network is greatly improved when new failure data arrives. It mitigates the problem of overfitting and thus enhances the accuracy of software reliability prediction. To the best of our knowledge, this is the first framework of applying SVM to software reliability prediction. Our proposed prediction approach is tested using four real-time control and flight dynamic application data sets. The following sections describe the implementation of our approach.

2. SVM Learning in Function Approximation

Notation

x_i	n-dimensional input vector, $x_i \in \Re^n$
y_i	target output value, $y_i \in \Re$
ϕ	high-dimensional feature space mapping function
w	weights vector
b	bias term
R	regularized risk function
$\|\|w\|\|^2$	weights vector norm
C	regularization constant
ϵ	Vapnik's linear loss function with ϵ-insensitivity zone
ξ_i, ξ_i^*	slack variables
α_i, α_i^*	Lagrange multipliers
K	kernel function

2.1. *Estimation of real-valued functions*

The basic idea of SVM for function approximation is mapping the data x into a high-dimensional feature space by a nonlinear mapping and then performing a linear regression in this feature space.[21] Assume that a total of l pairs of training patterns are given during SVM learning process,

$$(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_l, y_l)$$

where the inputs are n-dimensional vectors $x_i \in \Re^n$, and the target outputs are continuous values $y_i \in \Re$. The SVM model used for function approximation is:

$$f(x) = w \cdot \phi(x) + b \tag{1}$$

where $\phi(x)$ is the high-dimensional feature space that is nonlinearly mapped from the input space x. Thus, a nonlinear regression in the low-dimensional input space is transferred to a linear regression in a high-dimensional feature space.[21] The coefficients w and b can be estimated by minimizing the following regularized risk function R[18,21–24]:

$$R = \frac{1}{2}||w||^2 + C\frac{1}{l}\sum_{i=1}^{l} |y_i - f(x_i)|_\epsilon \tag{2}$$

where

$$|y_i - f(x_i)|_\epsilon = \begin{cases} 0 & \text{if } |y_i - f(x_i)| \le \epsilon, \\ |y_i - f(x_i)| - \epsilon & \text{otherwise.} \end{cases} \tag{3}$$

$||w||^2$ is the weights vector norm, which is used to constrain the model structure capacity in order to obtain better generalization performance. The second term is the Vapnik's linear loss function with ϵ-insensitivity zone as a measure for empirical error. The loss is zero if the difference between the predicted and observed value is less than or equal to ϵ. For all other cases, the loss is equal to the magnitude of the difference between the predicted value and the radius ϵ of ϵ-insensitivity zone. C is the regularization constant, representing the trade-off between the approximation error and the model structure. ϵ is equivalent to the approximation accuracy requirement for the training data points. Further, two positive slack variables ξ_i and ξ_i^* are introduced. We have

$$|y_i - f(x_i)| - \epsilon = \begin{cases} \xi & \text{for data "}\textit{above}\text{" an } \epsilon \text{ tube,} \\ \xi^* & \text{for data "}\textit{below}\text{" an } \epsilon \text{ tube.} \end{cases} \tag{4}$$

Thus, minimizing the risk function R in Eq. (2) is equivalent to minimizing the objective function R_{w,ξ,ξ^*}.

$$R_{w,\xi,\xi^*} = \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{5}$$

subject to constraints

$$\begin{cases} y_i - w \cdot \phi(x_i) - b \leq \epsilon + \xi_i & i = 1, \ldots, l, \\ w \cdot \phi(x_i) + b - y_i \leq \epsilon + \xi_i^* & i = 1, \ldots, l, \\ \xi_i, \xi_i^* \geq 0 & i = 1, \ldots, l. \end{cases} \tag{6}$$

This constrained optimization problem is typically solved by transforming into the dual problem, and its solution is given by the following explicit form:

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i, x) + b. \tag{7}$$

2.2. *Lagrange multipliers*

In Eq. (7), α_i and α_i^* are the Lagrange multipliers with $\alpha_i \times \alpha_i^* = 0$ and $\alpha_i, \alpha_i^* \geq 0$ for any $i = 1, \ldots, l$. They can be obtained by maximizing the following form:

$$-\epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) \tag{8}$$

subject to constraints

$$\begin{cases} \displaystyle\sum_{i=1}^{l}\alpha_i^* = \sum_{i=1}^{l}\alpha_i \\ 0 \leq \alpha_i, \alpha_i^* \leq C \quad i = 1, \ldots, l. \end{cases} \tag{9}$$

After learning, only some of coefficients $(\alpha_i - \alpha_i^*)$ in Eq. (7) differ from zero, and the corresponding training data points are referred to as support vectors. It is obvious that only the support vectors can fully decide the decision function in Eq. (7).

2.3. *Kernel function*

In Eq. (7), $K(x_i, x)$ is defined as the kernel function, which is the inner product of two vectors in feature space $\phi(x_i)$ and $\phi(x)$. By introducing the kernel function, we can deal with the feature spaces of arbitrary dimensionality without computing the mapping relationship $\phi(x)$ explicitly.[19] Some commonly used kernel functions are polynomial kernel function and Gaussian kernel function.

3. Implementation of Dynamic Software Reliability Prediction

The proposed software reliability prediction system shown in Fig. 1 consists of a failure history database and an iteratively and dynamically updated SVM learning-predicting process. When a software failure, x_i, occurs, the failure history database is updated and the accumulated failure data $(x_1, x_2, \ldots, x_i)$ is made available to the SVM learning process. The number of failure data increases over time during a dynamic system. Accordingly, the SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure sequence. After the SVM learning process is complete based on the currently available history failure data, next-step failure information, $\widehat{x}_{i+1}$, will be predicted.

3.1. *Formulation of the SVM-predictor*

In our proposed approach, unlike the existing mapping characteristics, we model the inter-relationship among software failure time data. More specifically, the input-output pattern fed into the network is the failure temporal sequence. The SVM learning scheme is applied to the failure time data, forcing the network to learn and recognize the inherent internal temporal property of software failure sequence. For one-step-ahead prediction, the input sequence and the desired output sequence

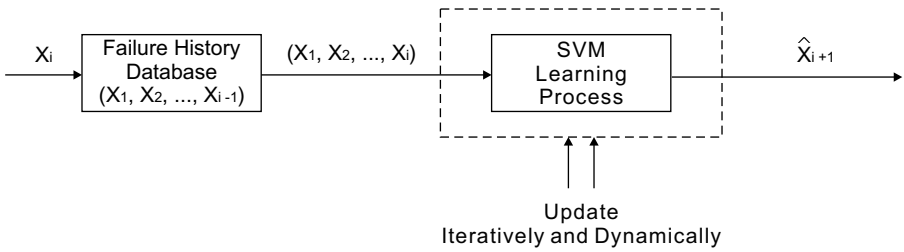

Fig. 1. Dynamic software reliability prediction framework.

should have one step delay during the learning process. The desired objective is to force the network to recognize the one-step-ahead temporal pattern. A sample input sequence and the corresponding one-step-ahead desired output sequence is defined as:

$$\begin{aligned} &\textit{Input Sequence}: && x_0, x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots \\ &\textit{Output Sequence}: && x_1, x_2, \ldots, x_i, x_{i+1}, x_{i+2}, \ldots \end{aligned}$$

where x_i is the failure time of the ith failure in the learning process. Once the network is trained based on all the currently available history failure data using the SVM learning procedure described in Sec. 2, the one-step-ahead failure time will be predicted. For comparison purposes, we also study the long-term prediction performance by using x_i as input and x_{i+d} as target output, where d ranges from 2 to 5.

Although our proposed software reliability growth modeling approach was originally intended for using time-domain data (actual failure time) as input to make predictions, if it is assumed that the data collected are interval-domain data, it is possible to develop new models by changing the input-output pair of the network without altering the inner architecture. Further, recent studies show that using testing time as the only influencing factor may not be appropriate for predicting software reliability.[25,26] Some environmental factors should be integrated. Examples of related environmental factors are program complexity, programmer skills, testing coverage, level of test-team members and reuse of existing code, etc.[25,27] Our proposed modeling approach is flexible to incorporate the related environmental factors by changing the scalar input to a vector input, if the values of the environmental factors corresponding to either failure time data or failure count data exist.

3.2. *Scaling of the input and output data*

All the inputs and outputs of the SVM network are scaled and normalized within the range of [0.1, 0.9] to minimize the impact of absolute scale. For this purpose, the actual values are scaled using the following relationship[28]:

$$y = \frac{0.8}{\Delta} x + \left(0.9 - 0.8 \times \frac{x_{\max}}{\Delta}\right) \tag{10}$$

where, y is the scaled value we feed into our network, x is the actual value before scaling, $x_{\max}$ is the maximum value in the samples. $x_{\min}$ is the minimum value among all the samples, and Δ is defined as $(x_{\max} - x_{\min})$.

After the training process, we test the prediction performance by scaling back all the network outputs to their actual values using the following equation:

$$x = \frac{y - 0.9}{0.8} \times \Delta + x_{\max} \tag{11}$$

4. Experimental Results

The performance of our proposed approach is tested using the same real-time control application and flight dynamic application data sets as cited in Park *et al.*[6] and Karunanithi *et al.*[4] We choose a common baseline to compare our results with related work cited in the literature. All four data sets used in the experiments are summarized as follows:

DATA-2: Real-time command and control application consisting of 21,700 assembly instructions and 136 failures.
DATA-11: Flight dynamic application consisting of 10,000 lines of code and 118 failures.
DATA-12: Flight dynamic application consisting of 22,500 lines of code and 180 failures.
DATA-13: Flight dynamic application consisting of 38,500 lines of code and 213 failures.

4.1. *Performance metrics*

When testing a proposed model, it is necessary to quantify its prediction accuracy in terms of some meaningful measures. The following statistical metrics are used for comparing prediction performance, namely, Next-Step-Predictability represented by Relative Prediction Error (RE), and Average Relative Prediction Error (AE).

$$\mathrm{RE} = \left| \frac{\hat{x}_i - x_i}{x_i} \right| \tag{12}$$

$$\mathrm{AE} = \frac{1}{(n - i_{\min} + 1)} \sum_{i=20}^{n} \left| \frac{\hat{x}_i - x_i}{x_i} \right| \times 100 \tag{13}$$

where $\hat{x}_i$ is the predicted value of failure time, and x_i is the actual value of failure time. $20 \leq i \leq n$, and n is the number of failure time data accumulated in real-time. Next-Step-Predictability is obtained by the percentage of the one-step-ahead predicted values fall within a pre-determined range of RE compared to their actual observed values. The larger the value of Next-Step-Predictability, or the smaller the value of AE, the closer are the predicted values to the actual values.

4.2. *Test results*

The results of the Next-Step-Predictability represented by relative prediction error (RE) using the four data sets are shown in Table 1. For example, using DATA-12, 95.63% of the next-step predicted values fall within 5% of their actual observed values. The results show that our proposed SVM predicting approach provides highly accurate prediction capability.

Table 2 summarizes the results of modeling the temporal inter-relationship among software failure time sequence using our proposed SVM approach. We use the same data sets as cited in Park *et al.*[6] and Karunanithi *et al.*[4] in order to

Table 1. Performance results of next-step-predictability.

Next-Step-Predictability (RE $\leq$ 5%)			
DATA-2	DATA-11	DATA-12	DATA-13
87.07%	93.88%	95.63%	95.31%

Table 2. Comparison of average relative prediction error (AE%) for next-step prediction.

Data Sets	Proposed SVM Approach	FFNN (Ref. 6)	RNN (Ref. 4)	FFNN (Ref. 4)
DATA-2	2.44	2.58	2.05	2.50
DATA-11	1.52	3.32	2.97	5.23
DATA-12	1.24	2.38	3.64	6.26
DATA-13	1.20	1.51	2.28	4.76

establish a common baseline for comparison purposes. Park *et al.*[6] applied failure sequence number as input and cumulative failure time as desired output in feed-forward neural network (FFNN). Based on the input-output learning pair of cumulative execution time and the corresponding accumulated number of defects disclosed, Karunanithi *et al.*[4] employed both feed-forward neural network (FFNN) and recurrent neural network (RNN) structures to model the failure process. These results are also summarized in Table 2. For example, using our proposed approach with data set DATA-12, the average relative prediction error (AE) is 1.24%. This error is lower than the results obtained by Park *et al.*[6] (2.38%) using feed-forward neural network, Karunanithi *et al.*[4] (3.64%) using recurrent neural network, and Karunanithi *et al.*[4] (6.26%) using feed-forward neural network. In all four data sets, the next-step prediction results show that using our proposed SVM approach yields a lower average relative prediction error compared to other neural network approaches, and is easily implemented to predict failures dynamically. Figures 2–5 show the predicted and actual values of the failure time for each data set in both short-term ($d = 1$) and long-term ($2 \leq d \leq 5$) situations.

5. Conclusion

In this paper, a novel support vector machine modeling approach for dynamic software reliability prediction based on software failure time data is proposed. Unlike traditional modeling approaches, we model the inter-relationship among software failure time data. The SVM learning scheme is applied to the failure time data, forcing the network to learn and recognize the inherent internal temporal property of software failure sequence. Further, the SVM learning process is iteratively and dynamically updated after every occurrence of new failure time data in order to capture the most current feature hidden inside the software failure sequence. Experimental results show that our proposed approach adapts well across different software projects, and has a better performance with respect to

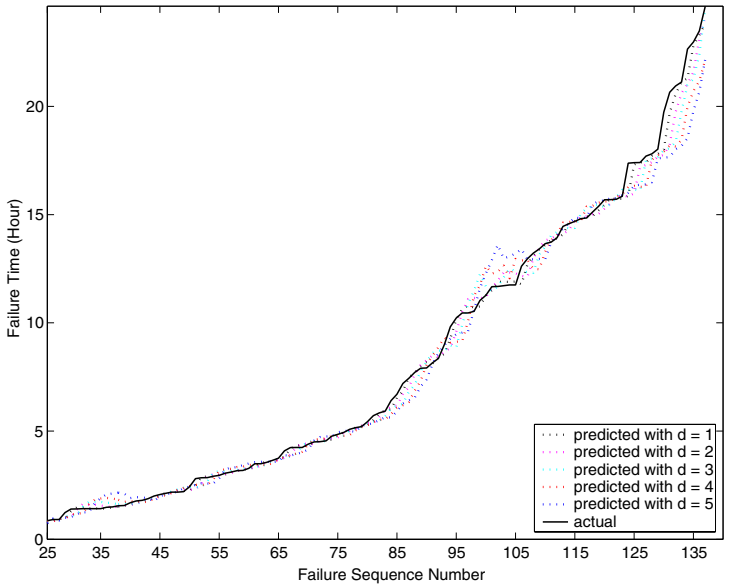

Fig. 2. Prediction performance using DATA-2.

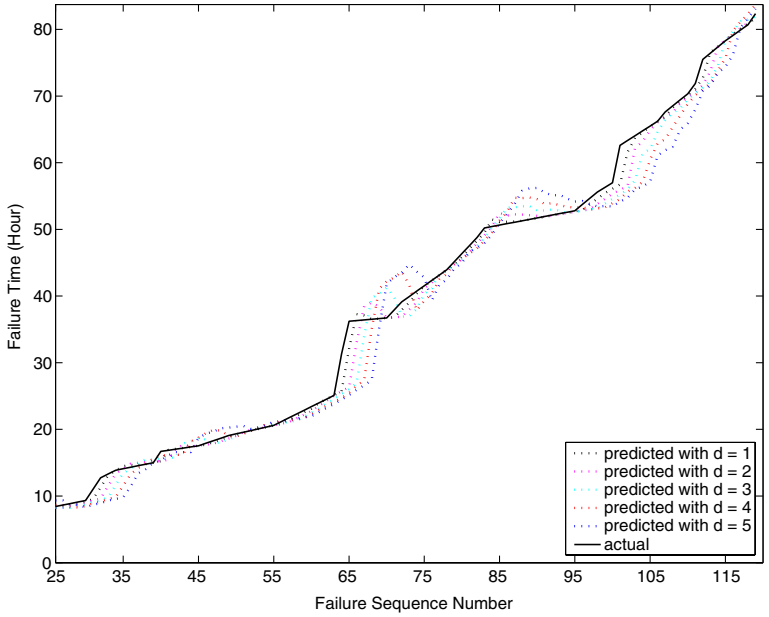

Fig. 3. Prediction performance using DATA-11.

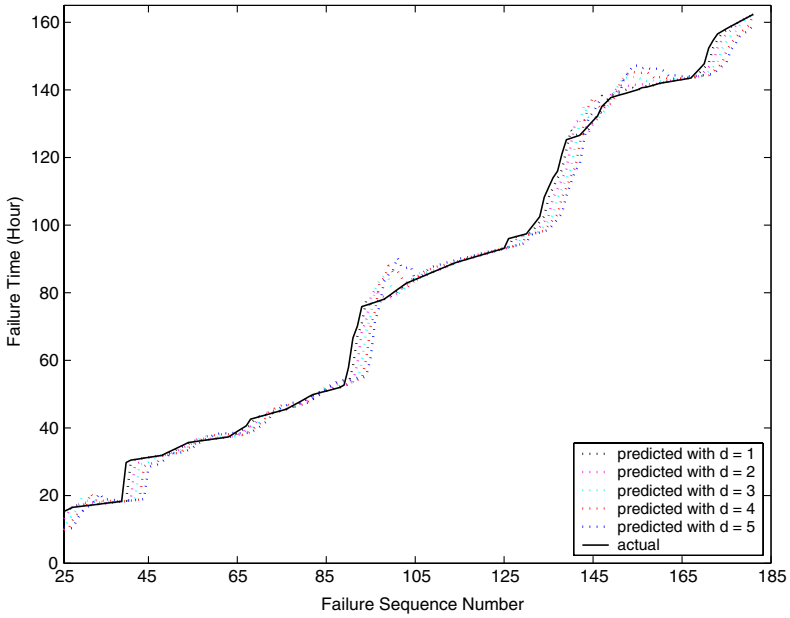

Fig. 4. Prediction performance using DATA-12.

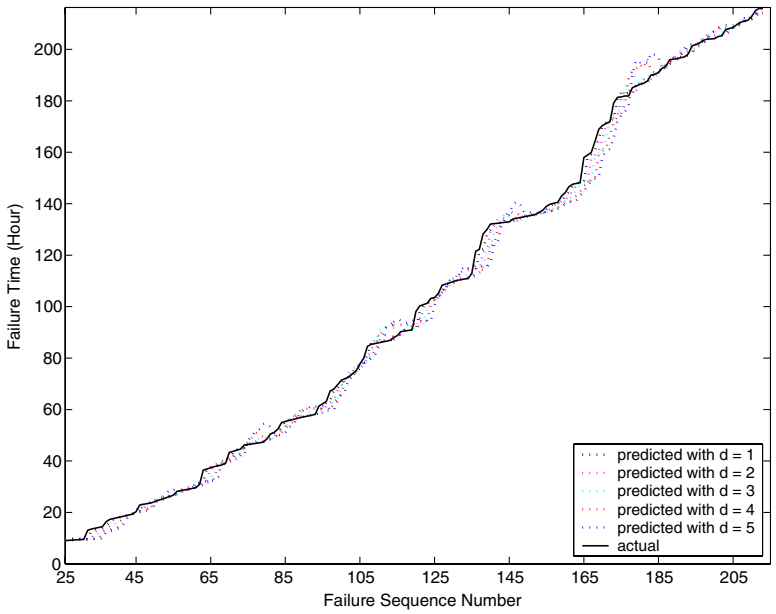

Fig. 5. Prediction performance using DATA-13.

the Next-Step-Predictability compared to the existing neural network modeling approaches.

Acknowledgments

This research was sponsored in part by a US DOE/EPSCoR West Virginia State Implementation Award through the Advanced Power & Electricity Research Center at West Virginia University. The authors would like to thank the reviewers for their helpful comments, and Drs. J. Y. Park, N. Karunanithi, Y. K. Malaiya, and D. Whitley for providing data sets DATA-11, DATA-12, DATA-13.

References

1. J. D. Musa, *Software Reliability Engineering* (McGraw-Hill, New York, 1998).
2. K. Y. Cai, C. Y. Wen and M. L. Zhang, A critical review on software reliability modeling, *Reliability Engineering and System Safety* **32**(3) (1991) 357–371.
3. K. Y. Cai, L. Cai, W. D. Wang, Z. Y. Yu and D. Zhang, On the neural network approach in software reliability modeling, *J. Systems and Software* **58**(1) (2001) 47–62.
4. N. Karunanithi, D. Whitley and Y. K. Malaiya, Prediction of software reliability using connectionist models, *IEEE Trans. Software Eng.* **18**(7) (1992) 563–574.
5. N. Karunanithi, D. Whitley and Y. K. Malaiya, Using neural networks in reliability prediction, *IEEE Software* **9**(4) (1992) 53–59.
6. J. Y. Park, S. U. Lee and J. H. Park, Neural network modeling for software reliability prediction from failure time data, *J. Electrical Engineering and Information Science* **4**(4) (1999) 533–538.
7. L. V. Utkin, S. V. Gurov and M. I. Shubinsky, A fuzzy software reliability model with multiple-error introduction and removal, *Int. J. Reliability, Quality and Safety Engineering* **9**(3) (2002) 215–227.
8. F. H. F. Leung, H. K. Lam, S. H. Ling and P. K. S. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, *IEEE Trans. Neural Networks* **14**(1) (2003) 79–88.
9. N. Karunanithi and Y. K. Malaiya, The scaling problem in neural networks for software reliability prediction, in *Proc. 3rd Int. Symp. Software Reliability Engineering* (Research Triangle Park, NC, 1992).
10. W. A. Adnan and M. H. Yaacob, An integrated neural-fuzzy system of software reliability prediction, in *Proc. 1st Int. Conf. Software Testing, Reliability and Quality Assurance* (New Delhi, India, 1994).
11. W. A. Adnan, M. Yaakob, R. Anas and M. R. Tamjis, Artificial neural network for software reliability assessment, in *2000 TENCON Proc. Intelligent Systems and Technologies for the New Millennium* (Kuala Lumpur, Malaysia, 2000).
12. S. H. Aljahdali, A. Sheta and D. Rine, Prediction of software reliability: A comparison between regression and neural network non-parametric models, in *Proc. ACS/IEEE Int. Conf. Computer Systems and Applications* (Beirut, Lebanon, 2001).
13. S. H. Aljahdali, A. Sheta and D. Rine, Predicting accumulated faults in software testing process using radial basis function network models, in *Proc. ISCA 17th Int. Conf. Computers and Their Applications* (San Francisco, CA, 2002).
14. S. L. Ho, M. Xie and T. N. Goh, A study of the connectionist models for software reliability prediction, *Computers and Mathematics with Applications* **46**(7) (2003) 1037–1045.

15. R. Sitte, Comparison of software-reliability-growth predictions: Neural networks vs parametric-recalibration, *IEEE Trans. Reliability* **48**(3) (1999) 285–291.
16. L. Tian and A. Noore, Software reliability prediction using recurrent neural network with Bayesian regularization, *Int. J. Neural Systems* **14**(3) (2004) 165–174.
17. L. Tian and A. Noore, Evolutionary neural network modeling for software cumulative failure time prediction, *Reliability Engineering and System Safety* **87**(1) (2005) 45–51.
18. V. Vapnik, S. E. Golowich and A. Smola, Support vector method for function approximation, regression estimation and signal processing, *Advances in Neural Information Processing Systems 9, in Proc. 1996 Conference* (Denver, CO, 1996).
19. L. J. Cao and F. E. H. Tay, Support vector machine with adaptive parameters in financial time series forecasting, *IEEE Trans. Neural Networks* **14**(6) (2003) 1506–1518.
20. J. Robinson and V. Kecman, Combining support vector machine learning with the discrete cosine transform in image compression, *IEEE Trans. Neural Networks* **14**(4) (2003) 950–958.
21. V. N. Vapnik, An overview of statistical learning theory, *IEEE Trans. Neural Networks* **10**(5) (1999) 988–999.
22. F. E. H. Tay and L. J. Cao, Application of support vector machines in financial time series forecasting, *Omega* **29**(4) (2001) 309–317.
23. K. J. Kim, Financial time series forecasting using support vector machines, *Neurocomputing* **55**(1–2) (2003) 307–319.
24. V. N. Vapnik, *Statistical Learning Theory*, Chap. 10.8, Examples of SV Machines for Pattern Recognition (John Wiley & Sons, New York, 1998).
25. H. Pham, *Software Reliability*, Chap. 8, Software Reliability Models with Environmental Factors (Springer, 2000).
26. H. Pham. Software reliability and cost models: Perspectives, comparison, and practice, *European Journal of Operational Research* **149**(3) (2003) 475–489.
27. X. Zhang, M.-Y. Shin and H. Pham, Exploratory analysis of environmental factors for enhancing the software reliability assessment, *J. Systems and Software* **57**(1) (2001) 73–78.
28. L. H. Tsoukalas and R. E. Uhrig, *Fuzzy and Neural Approaches in Engineering*, Chap. 11. Practical Aspects of Using Neural Networks (John Wiley & Sons, New York, 1996).

About the Authors

Liang Tian received his Ph.D. in Computer and Information Science in August 2005 from West Virginia Univeristy. Dr. Tian's research interests include evolutionary computation, neural network optimization, time series prediction, support vector machines, fuzzy systems, and software reliability modeling and prediction. His research has been supported by NASA and the US Department of Energy.

Afzel Noore obtained his Ph.D. in Electrical Engineering from West Virginia University. He worked as a digital design engineer at Philips India. From 1996 to 2003, Dr. Noore served as the Associate Dean for Academic Affairs in the College of Engineering and Mineral Resources at West Virginia University. He is an Associate

Professor in the Lane Department of Computer Science and Electrical Engineering. His research interests include software reliability modeling, machine learning, fuzzy and neural systems, fault-tolerant computing, microelectronics and biometrics. His research has been funded by Westinghouse, GE, NSF, Electric Power Research Institute, the US Department of Energy, and the US Department of Justice.