예측필터를 이용한 소프트웨어 신뢰성 예측

박 중 양† · 이 상 운†† · 박 재 흥†††

요 약

대부분의 소프트웨어 신뢰성 모델들은 소프트웨어 사용과 소프트웨어 고장 발생 과정에 대한 가정에 기반을 두고 있다. 따라서 보편적으로 적용 가능한 소프트웨어 신뢰성 모델이 없는 실정이다. 보편적인 소프트웨어 신뢰성 모델을 개발하기 위해, 본 논문은 시간단위 고장 데이터에 대한 일반적 소프트웨어 신뢰성 예측 모델로서 예측 필터를 제안하고, 14개의 다른 소프트웨어 프로젝트로부터 얻은 고장 데이터를 분석하여 그 유용성을 검토하였다. 제안된 예측필터와 다른 신경망 모델들과 통계적 소프트웨어 신뢰성 성장 모델들과의 성능 비교는 평균 상대 예측 오차에 기반하였다. 그 결과 예측필터는 일반적으로 단순한 모델을 제공할 뿐만 아니라 다양한 소프트웨어 프로젝트에 적합함을 알 수 있었다.

Software Reliability Prediction Using Predictive Filter

Joong-Yang Park† · Sang-Un Lee†† · Jae-Heung Park†††

ABSTRACT

Almost all existing software reliability models are based on the assumptions of the software usage and software failure process. There, therefore, is no universally applicable software reliability model. To develop a universal software reliability model, this paper suggests the predictive filter as a general software reliability prediction model for time domain failure data. Its usefulness is empirically verified by analyzing the failure data sets obtained from 14 different software projects. Based on the average relative prediction error, the suggested predictive filter is compared with other well-known neural network models and statistical software reliability growth models. Experimental results show that the predictive filter generally results in a simple model and adapts well across different software projects.

1. Introduction

In recent years, there is a growing use of computer systems and software systems have been most important parts of many complex and critical computer systems. Since failures of a software system could produce severe consequences in terms of human life, environment impact or economical losses, software systems are required to be sufficiently reliable for their intended purposes. A software system is said to be reliable if it performs failure-free operation for a specific exposure period under a wide variety of usage environments and operations. To many customers, software reliability is one of the most important aspects of software quality. Measurement of software reliability includes reliability estimation and reliability prediction activities. A number of software reliability models has been developed for estimating and predicting the software reliability from the failure data obtained during testing and operational phases. Predictability of a software reliability model is the capability of the model to predict future behaviors from present and past failure behaviors. Ramamoorthy and Bastani [16] classifies software reliability models

† 정 회 원 : 경상대학교 통계학과 교수
†† 정 회 원 : 경상대학교 대학원 전자계산학과
††† 정 회 원 : 경상대학교 컴퓨터과학과 교수
논문접수 : 2000년 2월 15일, 심사완료 : 2000년 7월 8일

according to the development phases of software life-cycle, while Goel [6] divides them according to the nature of failure process into time-between-failures models, failure-count models, fault seeding models and input domain-based models. We consider software reliability models applicable in the testing and debugging phase. Since fault corrections are necessary in the testing and debugging phase, software reliability growth models (SRGMs) taking fault corrections into account are mainly used in this phase. Most of time-between-failures models and failure- count models belong to the class of SRGMs. Such SRGMs offer overall reliability assessment and prediction and are also used in determining an appropriate release time. SRGMs usually define the software reliability as

$$R_T = \Pr(\text{no failure occurs within } [0, T]),$$

where T is the exposure period whose time unit is the calendar or execution time. It is usually assumed that R_T follows a certain probability distribution, for example, $R_T = \exp\left(-\int_0^T z(t)dt\right)$, where $z(t)$ is the failure rate function. Once the failure rate function is estimated, the expected time to next failure and expected number of failures up to a certain specified time can be predicted and then used as reliability measures.

Almost all existing SRGMs are based on the assumptions with respect to the nature of software faults and the stochastic behavior of failures. They include several parameters to reflect on various assumptions on the software development and usage environment. This implies that there is no universally applicable SRGM which can be trusted in all circumstances [3]. Selection of a particular SRGM is thus very important in practical applications. Several approaches for selecting the best SRGM were suggested by Abdel- Ghaly, Chan and Littlewood [1] and Brocklehurst, Chan, Littlewood and Snell [3]. However, it is still desirable to develop software reliability models that do not require any assumptions with respect to the development and usage environment. If we have a system that develops its own model based on the past failure history of software system, such assumptions can be eliminated. Thus a new approach using neural networks has been introduced by Karunanithi, Whitley and Malaiya [7, 8]. The neural network requires only failure history as input and no assumptions are made a priori. Karunanithi, Whitley and Malaiya [7, 8] evaluated the predictive accuracy of feedforward networks (FFNs) and Jordan networks and then compared with several well-known statistical SRGMs for the failure count data. Park, Lee and Park [14] also performed a similar research for the failure time data.

In this paper we suggest a predictive filter for predicting software reliability based on the time domain failure data (both failure count data and failure time data). The proposed predictive filter is derived from the adaptive filter. The predictive filter is simple and its performance is better than those of the neural networks mentioned in the previous paragraph. Section 2 first briefly reviews some related works. The predictive filter and corresponding training regimes are presented in Section 3. Section 4 investigates its performance empirically and compares with other neural network models and SRGMs. Performance evaluations and comparisons are conducted by computing the next-step prediction error. Finally conclusions are presented in Section 5.

2. Related Works and Motivation

An important feature of neural networks is the ability to learn from their environment and through such learning improvement in performance to some extent is achieved. (Refer to Lippman [9] for currently available neural network models and learning procedures.) Thus neural networks have been applied to parameter estimation and prediction of future outcomes. Two well-known classes suitable for prediction are FFNs and recurrent networks. However, only a few applications of neural networks to software reliability prediction have appeared in the literature. Karunanithi, Whitley and Malaiya [7, 8] modeled software reliability growth with neural networks. They have shown that FFNs and partial recurrent networks (Elman and Jordan net-

works) can be applied to software reliability prediction.

Suppose that pairs $(t_i, m_i), i=1,2,\cdots, n$ are gathered from the software testing, where m_i denotes the cumulative number of failures found up to time t_i. A data consisting of pairs (t_i, m_i) is called the failure count data. Karunanithi, Whitley and Malaiya [7, 8] studied FFNs and partial recurrent networks for predicting m_{l+p}, $p=1,2,\cdots$ assuming that (t_i, m_i), $i=1,2,\cdots, l$ are available for training. The following two training regimes were considered.

(1) Generalization training : each input t_i is associated with target m_i.

(2) Prediction training : each input t_{i-1} is associated with target m_i.

The predictive accuracy of FFNs and partial recurrent networks were evaluated and then compared with those of several well-known statistical SRGMs. The data sets collected from 14 different software systems were used for evaluating and comparing the predictive accuracy. Brief description of the 14 data sets (Referred to as Data1, Data2, ··· in Karunanithi, Whitley and Malaiya [8] and herein.) is reproduced in <Table 1>. They have shown that FFNs and partial recurrent networks adapt well across different data sets and exhibit a better end-point predictive accuracy. However, they were unable to make any strong conclusions about the relative advantages for next-step predictions.

The observation time intervals, $t_i - t_{i-1}$, are variable for Data4 and Data5, but constant for other data sets. Another noteworthy point is that Data2, Data11, Data12, and Data13 consist of the pairs (i, s_i), $i=1,2,\cdots$ where i is the failure sequence number and s_i is the ith failure time. Such data sets are called the failure time data. Karunanithi, Whitley and Malaiya [7, 8] presented the failure times as input to the network and then associated them with corresponding failure sequence number as target. In the case of failure time data, the failure times are the observations of stochastic failure process. It is not reasonable to analyze and predict the failure sequence numbers. What we need to predict is not the failure sequence number but rather the stochastic process producing the failure times. Therefore, the above two training regimes are not appropriate for the failure time data. Park, Lee and Park [15] considered FFNs and 4 training regimes for the failure time data.

<Table 1> Brief Description of 14 Data Sets

Data Set	Reference	System Size in LOC	Failure Category	m_n	n	Application
Data 1	[11]	1,000	Failure Count Data	27	14	Class Compiler Project
Data 2	[12, 13]	21,700	Failure Time Data	136	136	Realtime Control
Data 3	[14]	40,000	Failure Count Data	46	21	On-line Data Entry
Data 4	[14]	1,317,000	Failure Count Data	328	17	Database Application
Data 5	[14]	35,000	Failure Count Data	279	10	Hardware Control
Data 6	[17]	240,000	Failure Count Data	3,207	13	Application Software
Data 7	[19]	870,000	Failure Count Data	535	109	Realtime Control
Data 8	[18]	200,000	Failure Count Data	481	111	Monitoring and Realtime Control
Data 9	[18]	14,5000	Failure Count Data	55	199	Railway Interlocking
Data 10	[18]	90,000	Failure Count Data	198	16	Monitoring and Realtime Control
Data 11	[2]	10,000	Failure Time Data	118	118	Flight Dynamic
Data 12	[2]	22,500	Failure Time Data	180	180	Flight Dynamic
Data 13	[2]	38,500	Failure Time Data	213	213	Flight Dynamic
Data 14	[19]	not known	Failure Count Data	266	46	Realtime Control

(1) Each input i is associated with target x_i.

(2) Each input i is associated with target x_{i+1}.

(3) Each input i is associated with target s_i.

(4) Etach input i is associated with target s_{i+1}.

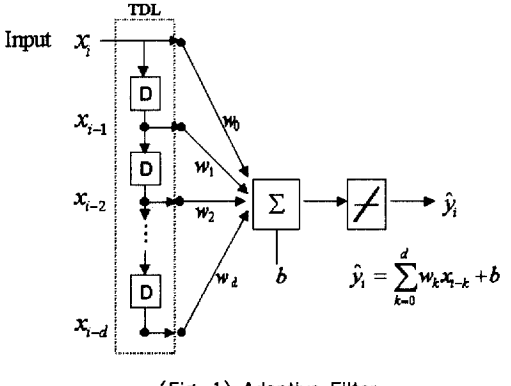
Here $x_i = s_i - s_{i-1}$ for $i=1,2,\cdots$. Training regimes (1) and (3) are for generalization, while training regimes (2) and (4) are for prediction. It was empirically shown that FFNs with training regimes (3) and (4) work well

for software reliability prediction and their predictive capability is not influenced much by the training data set size. The suggested approach was shown to be robust to outliers. However, more hidden neurons are required than the approach considered by Karunanithi, Whitley and Malaiya [6, 7].

A good predictive model should predict future behavior well, compute useful quantities, and be simple and widely applicable. One of such neural networks is the adaptive filter. The adaptive filter is simple and expected to provide good next-step prediction. It has been known for following two outstanding features : the ability to learn systems with unknown parameters and signals with unknown statistics, and the ability to track an environment which is varying with time. Refer to Widrow and Sterns [20] for details of the adaptive filter. We will consider the adaptive filter as a candidate network for software reliability prediction in the next section.

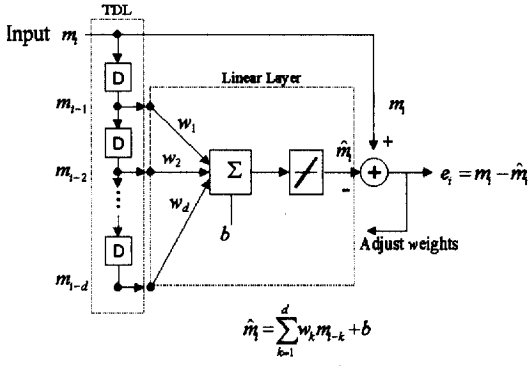
3. Predictive Filter for Software Reliability Prediction

An adaptive filter is created by combining a tapped delay line (TDL) with an ADALINE (Adaptive Linear Neuron Network) [5]. The ADALINE network is similar to the perceptron, but their transfer function is linear rather than hard-limiting. This network can only solve linearly separable problems. To solve a nonlinear problems, we need a new component, the TDL, to make full use of the ADALINE network. The adaptive filter can automatically adapt in the face of environment changes. (Fig. 1) shows a simple description of the adaptive filter, in which x_i is the input signal at time i, $\hat{y}_i = \sum_{k=0}^{d} w_k x_{i-k} + b$ is the corresponding output, d is the number of delays, w_k's are adaptive weights, and b is the bias (or threshold).



(Fig. 1) Adaptive Filter

This network is usually referred to in the digital signal processing field as a finite impulse response (FIR) filter (also known as input-delay neural network). This architecture has been used quite successfully in a number of practical applications including speech recognition and time series prediction [4]. If we adjust the adaptive weights in a way that the corresponding output $\hat{y}_i$ matches as best as possible a desired signal (target) y_i, the adaptive filter can predict the future values of the signal. We thus use the network described in (Fig. 2) for the failure count data. This network may be called the predictive filter, which is a modified adaptive filter. Specific modifications are : (1) m_i is not connected to the linear neuron filter ; (2) errors are computed and backpropagated to adjust connection weights.



(Fig. 2) Predictive Filter

The cumulative number of failures found up to t_i, is then predicted by $\hat{m}_i = \sum_{k=1}^{d} w_k m_{i-k} + b$, which is a linear combination of m_{i-k}, $k = 1, 2, \cdots, d$. This implies

that the predictive filter primarily aims the next-step prediction. However, if we use $\hat{m}_i$ as the input signal at time i and m_{i-k}'s as the corresponding delayed signal s, we can obtain $\hat{m}_{i+1}$ from the predictive filter. Repetition of this procedure enables us to predict $\hat{m}_{i+k}$, $k=0,1,\cdots$ at time $i-1$. At this time we should note that the observation time intervals, t_i-t_{i-1}, are required to be constant for the application of the above predictive filter. On the other hand, we need to replace m_i in (Fig. 2) with s_i for applying the predictive filter to the failure time data. That is, $\hat{s}_i=\sum_{k=1}^{d} w_k s_{i-k}+b$ and $e_i=s_i-\hat{s}_i$.

4. Prediction Experiments and Results

This section empirically studies predictive accuracy of the predictive filter suggested in the previous section. In order to provide reasonable comparisons and analyses on the predictive accuracy of various models, it is necessary to use as many data sets as possible. Finding a large collection of data sets from different software projects is not an easy task because many software companies consider their products' failure history as a classified record. The data sets described in Section 2, which were collected from various software systems and development environments, are used in our experiment. Data1, Data3-10, and Data14 are the failure count data and Data2, and Data11-13 are the failure time data.

In comparing different models, it is necessary to quantify their prediction accuracy in terms of some meaningful measures. Three distinct approaches that are very common in software reliability research community are goodness-of-fit, next-step predictability, and variable-term predictability [8, 10]. Using these approaches, a two-component predictability measure consisting of average relative prediction error (AE) and average bias (AB) was proposed by Malaiya, Karunanithi, and Verma [10]. AE is a measure of how well a model predicts through the test phase, and AB is the general bias of the model. In this paper, we use the AE measure for next-step predictability. For 14 different software projects, the next-step AE of predictive filter is computed and compared with those of other well-known neural networks and SRGMs simulated by Karunanithi, Whitley, and Malaiya [8] and Park, Lee, and Park [15].

4.1 Case A : Failure Count Data

Generally, we assume that the data set size is sufficiently large when its size is larger than 30 (referred to statistical field). Data set size, n, of Data1, Data3-6, and Data10 are not large enough for the neural network experiments. Furthermore the observation time intervals t_i-t_{i-1} of Data4 and Data5 are not constant. Therefore, the predictive accuracy of the predictive filter evaluated for these data sets is expected to be low. Nevertheless, we also performed experiments for these data sets for the sake of reference.

Let l denote the training data set size. And r is the prediction distance. For example, $r=1$ for the next-step prediction and $r=(n-l)$ for the end-point prediction. For given number of delays d, the next-step prediction experiments proceed as follows :

Step (1) Set $l=d$.

Step (2) Train each neural network model by using the LMS (Least Mean Square error) algorithm and the training data set $m_1, m_2, \cdots, m_l$.

Step (3) For each trained network, predict j step ahead cumulative number of failures $\hat{m}_{l+j}$ for $j=1,2,\cdots,r$ and then compute corresponding relative prediction errors,

$$e_{lj}=(\hat{m}_{l+j}-m_{l+j})/m_{l+j}\cdot 100.$$

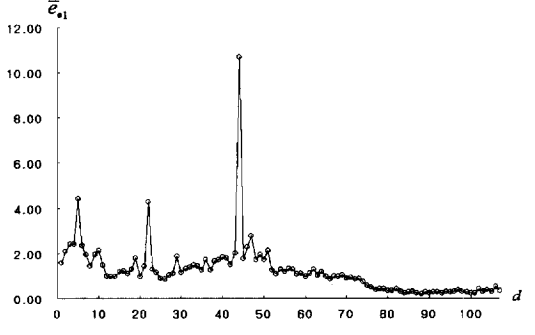
Step (4) Increase l by 1 and repeat Steps (2) and (3) until $l=n-r$.

Step (5) Compute the average of e_{lj} over l.

The average of e_{lj} over l is denoted by $\bar{e}_{\cdot j}$ and referred to as the AE. Karunanithi, Whitley and Malaiya [7, 8] considered the averages of e_{l1} and $e_{l(n-l)}$ over

l as the predictability measures, which correspond to the next-step and end-point prediction errors, respectively. Since the predictive filter primarily aims the next-step prediction, we will use $\bar{e}_{\cdot 1}$. For SRGMs $\hat{m}_{l+j}$ is the estimate of the expected number of failures up to t_{l+j} obtained from m_j, $j=1,2,\cdots,l$. Then e_{lj} and $\bar{e}_{\cdot 1}$ for SRGMs can be similarly obtained.

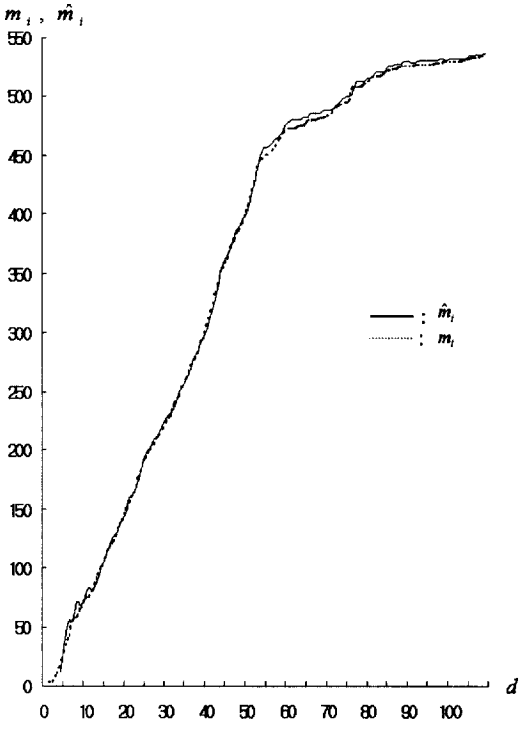
The most critical problem in implementing the predictive filter is to determine the appropriate value of d. To find a suitable value of d, we follow a trial-and-error approach. That is, Steps (1)-(5) are performed for $d=0, 1, \cdots$ and the resulting $\bar{e}_{\cdot 1}$'s are plotted against d. An appropriate value of d is selected by examining the plot. For example, (Fig. 3) shows the plot of $\bar{e}_{\cdot 1}$ against d for Data7.



(Fig. 3) Plot of $\bar{e}_{\cdot 1}$ against d for Data7

Examining (Fig. 3), we find that $\bar{e}_{\cdot 1}$ achieves its minimum when $d > 70$. However, it is apparent that overfitting occurs when d is too large. We thus consider the cases where d is small. $\bar{e}_{\cdot 1}$'s for $d=1,8, 11-18, \cdots$ are practically comparable to each other. Since a simpler model is usually desirable, we select 1 as an appropriate value of d. (Fig. 4) shows m_i and $\hat{m}_i$ for $d=1$. <Table 2> shows selected values of d for the 10 failure count data sets. Using the values of d in <Table 2>, Steps (1)-(5) were performed for each data set.

The resulting $\bar{e}_{\cdot 1}$'s were presented and compared with those of Karunanithi, Whitley and Malaiya [8] in <Table 3>. The numbers in parentheses are the corresponding ranks.



(Fig. 4) Plot of m_i and $\hat{m}_i$ for Data7 ($d=1$)

<Table 2> Selected values of d for 10 failure count data sets

Data Set	Data 1	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 14
Number of Delays	1	1	4	2	1	1	1	1	3	1

Experiment results for the failure count data suggest the followings :

(1) The predictive filter performs well when the data set is large. (Data7-9) That is, the predictive filter is efficient for software reliability prediction especially when the data set size is large ;

(2) When the data set size is small, all models seem to have almost similar next-step prediction accuracy. The neural network models and SRGMs

are either too pessimistic or too optimistic. Therefore, there is no model that works best for all software projects when the data set is small.

(3) The number of hidden units in the neural network models simulated by Karunanithi, Whitley and Malaiya [8] varied from 0 to 4 depending on the size of the training data set and the nature of the data set. But in the predictive filter, the number of delays is 1 for 7 data sets out of 10 data sets and does not exceed 4. That is, one or two delays is enough for good predictability ;

(4) Most of the SRGMs have only 2 or 3 parameters. The number of parameters in the predictive filter is $d+1$. Since $1 \le d \le 4$ for the 10 data sets, the number of parameters for good prediction is almost the same with those for SRGMs.

〈Table 3〉 $\bar{e}_{\cdot 1}$ for 10 Failure Count Data Sets

Model		Next-step Average Prediction Error (Rank)									
		Data 1	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 14
Neural Networks		7.53 (6)	10.56 (10)	6.27 (7)	8.48 (5)	4.47 (7)				9.59 (10)	3.53 (5)
	FFN-Generalization	9.37 (9)	8.44 (9)	5.28 (2)	10.00 (8)	4.33 (6)	3.66 (9)	6.56 (8)	11.82 (9)	5.90 (7)	4.56 (8)
	FFN-Prediction	5.61 (2)	6.84 (5)		6.95 (4)	4.51 (8)	3.74 (10)	5.24 (7)	6.72 (3)	7.45 (9)	4.80 (9)
	JordanNet-Generalization	6.79 (4)	6.84 (5)	8.84 (9)		5.25 (10)	2.97 (6)	9.11 (10)	9.72 (5)	4.08 (4)	4.86 (10)
	JordanNet-Prediction		6.03 (2)	6.11 (5)	8.67 (6)	5.24 (9)	2.90 (4)	3.73 (3)	6.41 (2)		4.50 (7)
Statistical SRGMs	Logarithmic	7.33 (5)	7.78 (7)	5.93 (3)	6.42 (3)		2.88 (3)	4.47 (6)	10.20 (6)	3.75 (3)	3.24 (3)
	Inverse Polynomial	9.21 (8)	6.17 (3)	7.95 (8)	9.71 (7)	3.59 (4)	2.92 (5)	4.45 (5)	9.09 (4)	4.90 (5)	
	Exponential	6.67 (3)	7.85 (8)	6.01 (4)	6.15 (2)	3.51 (2)	2.50 (2)	4.26 (4)	13.06 (10)	3.28 (2)	3.52 (4)
	Power	11.15 (10)	6.55 (4)	9.40 (10)	23.36 (10)	3.51 (2)	3.19 (8)	7.06 (9)	11.36 (8)	5.53 (6)	3.20 (2)
	Delayed S-shape	8.03 (7)		6.25 (6)	10.90 (9)	3.63 (5)	3.07 (7)	3.39 (2)	10.86 (7)	6.57 (8)	3.55 (6)

4.2 Case B : Failure Time Data

We now perform prediction experiment for the failure time data. For given number of delays d, the next-step prediction experiments proceed as follows :

Step (1) Set $l = d$.

Step (2) Train each neural network model by using the LMS algorithm and the training data set $s_1, s_2, \cdots, s_l$.

Step (3) For each trained network, predict j step ahead cumulative number of failures $\hat{s}_{l+j}$ for $j = 1, 2, \cdots, r$ and then compute corresponding relative prediction errors,

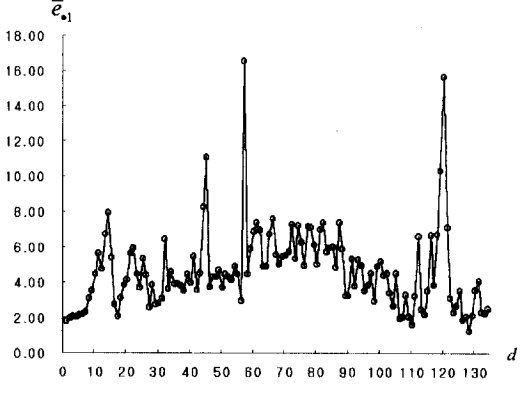
$$e_{lj} = (\hat{s}_{l+j} - s_{l+j}) / s_{l+j} \cdot 100 .$$

Step (4) Increase l by 1 and repeat Steps (2) and (3) until $l = n - r$.

Step (5) Compute the average of e_{lj} over l.

As discussed in the previous subsection, we use $\bar{e}_{\cdot 1}$ as the predictability measure. For SRGMs $\hat{s}_{l+j}$ is the estimate of the $(l+j)$ th failure time obtained from s_j, $j = 1, 2, \cdots, l$. Then e_{lj} and $\bar{e}_{\cdot 1}$ for SRGMs can be also obtained.

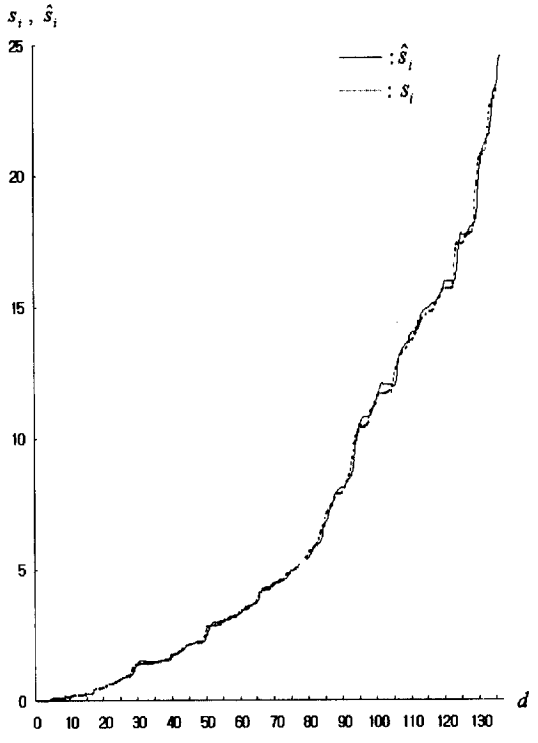
A suitable value of d is selected by examining the plot of $\bar{e}_{\cdot 1}$ against d as done in Section 4.1. For example, $\bar{e}_{\cdot 1}$ for Data2 are depicted in (Fig. 5).



(Fig. 5) Plot of $\bar{e}_{\cdot 1}$ against d for Data2

Apparently $d = 1$ is preferred to other values of d. (Fig. 6) shows s_i and $\hat{s}_i$ for $d = 1$. <Table 4> shows selected values of d for the 4 failure time data sets, each of which is either 1 or 2. Using the values of d in <Table 4>, the next-step predictions were performed for each failure time data set.

The resulting $\bar{e}_{\cdot 1}$'s were presented and compared with those of Park, Lee and Park [15] in <Table 5>.



(Fig. 6) Plot of s_i and $\hat{s}_i$ for Data2 ($d=1$)

〈Table 4〉 Selected values of d for 4 failure time data sets

Data Set	Data2	Data11	Data12	Data13
Number of Delays	1	2	2	1

〈Table 5〉 $\overline{e}_{\cdot 1}$ for 4 Failure Time Data Sets

Model		Next-step Average Prediction Error (Rank)			
		Data2	Data11	Data12	Data13
Neural Networks	Predictive Filter				1.94 (3)
	FFN -Generalization	2.28 (2)	3.32 (2)	2.28 (2)	1.58 (2)
	FFN -Prediction	2.58 (3)	3.32 (2)	2.38 (3)	

Experiment results for the failure time data suggest the followings :

(1) Predictive filter is consistently ranked in the top one ranks except for Data13. This observation corroborates that predictive filter is better suited for next-step prediction than other well-known neural network models.

(2) The number of delays is only one or two. For good prediction, the number of parameters in predictive filter is as small as those in SRGMs. On the contrary, Other neural network models, simulated by Park, Lee, and Park [15], require much more parameters since the number of hidden units is between 10 and 27.

In our experiments, only 4 failure time data set are used. For the sake of more general conclusion, experiments of more data set will be necessary. However, experiment results indicate that the predictive filter works well enough for software reliability prediction based on the failure time data.

5. Conclusions

As an attempt to develop a universal software reliability prediction model, this paper suggests the predictive filter for time domain failure data. We investigated its predictive performance by using 14 different software failure data. Based on the next-step AE, the suggested predictive filter is compared with other well-known neural network models and statistical software reliability growth models. Experimental results show that performance of the predictive filter is better than other neural network models and statistical SRGMs for large time domain failure data. (e.g., $n > 100$, except for Data13) Moreover, for small data sets, statistical SRGMs and neural network models do not show any significant difference. We thus come to the conclusion that the predictive filter is applicable as a general software reliability prediction model for the time domain failure data.

However, more data sets need to be gathered and analyzed for more general conclusions. Especially for the case where the failure time data set is involved.

The drawback of predictive filter can be apply in constant time interval data. Therefore, neural network

modeling for variable time interval data be necessary. Thus, our research in the future will be directed to this problem.

References

[1] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, "Evaluation of Competing]Software Reliability Predictions," IEEE Trans. Software Eng., Vol.SE-12, pp.950-967, 1986.

[2] B. M. Anna-Mary, "A Study of the Musa Reliability Model," M.S. dissertation, Univ. Maryland, 1980.

[3] S. Brocklehurst, B. Y. Chan, B. Littlewood and J. Snell, "Recalibrating Software Reliability Models," IEEE Trans. Software Eng., Vol.16, pp.458-470, 1990.

[4] D. S. Clouse, C. L. Giles, B. G. Horne, and G. W. Cottrell, "Time-Delay Neural Networks : Representation and Induction of Finite State Machines," IEEE Trans. on Neural Networks, Vol.8, No.5, pp. 1065-1070, 1997.

[5] H. Demuth and M. Beale, "*Neural Network Toolbox User's Guide, Ver. 3.0,*" Math Works, Inc. 1997.

[6] A. L. Goel, "Software Reliability Models : Assumptions, Limitations, and Applicability," IEEE Trans. on Software Eng., Dec. pp.1411-1423, 1985.

[7] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Prediction of Software Reliability Using Connectionist Models," IEEE Trans. Software Eng., Vol. 18, pp.563-574, 1992.

[8] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Using Neural Networks in Reliability Prediction," IEEE Software, Vol.18, pp.53-59, 1992.

[9] R. Lippman, "An Introduction to Computing with Neural Nets," IEEE Acoustics, Speech and Signal Processing, pp.4-22, 1987.

[10] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability Measures for Software Reliability Models," IEEE Trans. on Reliability, Vol.41, No.4, pp.539-546, 1992.

[11] K. Matsumoto, T. Inoue, T. Kikuno, and K. Torii, "Experimental Evaluation of Software Reliability Growth Models," Proc. IEEE Conf. FTCS-18, pp. 148-153, 1988.

[12] J. D. Musa, A. Lannino, and K. Okumoto, "*Software Reliability : Measurement, Prediction, Application,*" McGraw-Hill, New York, 1987.

[13] J. D. Musa, "Software Reliability Data," Technical Report, Data and Analysis Center for Software, Rome Air Development Center, Griffins AFB, New York, 1979.

[14] M. Ohba, "Software Reliability Analysis Models," IBM H. Res. Develop., Vol.28, pp.428-443, 1984.

[15] J. Y. Park, S. U. Lee, and J. H. Park, "Neural Network Modeling for Software Reliability Prediction from Failure Time Data," Journal of Electrical Eng. and Information Science, Vol.4, No.4, pp.533-538, 1999.

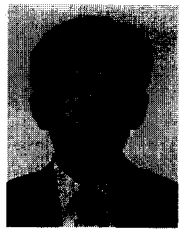
[16] C. V. Ramamoorthy and F. B. Bastani, "Software Reliability-Status and Perspectives," IEEE Trans. Soft. Eng., SE-8, No.4, July, pp.354-371, 1982.

[17] M. L. Shooman, "Probablistic Models for Software Reliability Prediction," Statistical Computer Performance Evaluation, New York Academic, pp.485-502, 1972.

[18] Y. Thoma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural Approach to the Estimation of the Number of Residual Software Faults Based on the Hyper-Geometric Distribution," IEEE Trans. on Software Eng., Vol.15, pp.345-355, 1989.

[19] Y. Thoma, H. Yamano, M. Ohba, and R. Jacoby, "Parameter Estimation of the Hyper-Geometric Distribution Model for Real Test/Debug Data," Dept. Computer Science, Tokyo Inst. Tech., Tech. REP.901002, 1990.

[20] B. Widrow and S. D. Sterns, "*Adaptive Signal Processing,*" Prantice-Hall, New York, 1985.



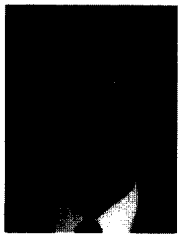
Joong-Yang Park

received his B.S. degree in Applied Statistics from Yonsei University in 1982, the M.S. and Ph.D. degrees in Industrial Engineering from Korea Advanced Institute of Science and Technology in 1984 and 1994.

Since 1985 he has been a professor of the Department of Statistics, Gyeongsang National University, Chinju, Korea.

His research interests are in the area of Software Reliability, Neural Networks, Linear Statistical Models and Experimental Designs.

e-mail : parkjy@nongae.gsnu.ac.kr

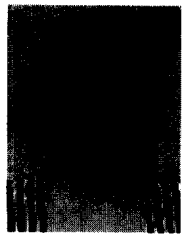


Sang-Un Lee

received his B.S. degree in Avionics for Hankuk Aviation University in 1987, the M.S. degree in Computer Science from Gyeongsang National University in 1997. He is currently working toward the Ph.D. degree in Computer Science at the Gyeongsang National University, Chinju, Korea.

His research interests are Software Quality Assurance and Reliability Modeling, Neural Networks.

e-mail : sangun_lee@hanmail.net



Jae-Heung Park

received his B.S. degree in Mathematics from Chungbuk National university in 1978, the M.S. and Ph.D. degrees in Computer Science from ChungAng National University in 1980 and 1988.

Since 1983 he has been a professor of the Department of Computer Science, Gyeongsang National University, Chinju, Korea.

His research interests are in the area of Software Reliability, Neural Networks, Automatic Testing Tool, System Analysis and Design.

e-mail : pjh@nongae.gsnu.ac.kr