Software Reliability Predictions using Artificial Neural Networks

Q.P. Hu, M. Xie and S.H. Ng

Department of Industrial and Systems Engineering
National University of Singapore, Singapore

8.1 Introduction

Computer-based artificial systems have been widely applied in nearly every field of human activities. Whenever people rely heavily on some product/technique, they want to make sure that it is reliable. However, computer systems are not as reliable as expected, and software has always been a major cause of the problems. With the increasing reliability of hardware and growing complexity of software, the software reliability is a rising concern for both developer and users. Software reliability engineering (SRE) has attracted a lot of interests and research in the software community and software reliability modeling is one major part of SRE research.

Software reliability modeling describes the fault-related behaviors of the software testing process and is one of the important achievements in software reliability research activities. The information provided by the models is helpful in making management decisions on issues regarding the software reliability. They have been successfully applied in practical software projects, such as cost-analysis [17, 34], testing-resource allocation [6, 37], test-stopping decision [21, 32] and fault-tolerance system analysis [11, 19].

Generally, software reliability models can be grouped into two categories: analytical software reliability growth models (SRGMs) and data-driven models. Analytical SRGMs use stochastic models to describe the software failure process under several assumptions to provide mathematical tractability [18, 22, 23, 24, 31]. The major drawbacks of these models are their restrictive assumptions. On the other hand, most data-driven models follow the approach of time series analysis, including both traditional autoregressive methods [5] and modern artificial neural network (ANN) techniques [15]. These models are developed from past software failure history data. Specially, ANNs are universal functional approxima-

Q.P. Hu et al.: *Software Reliability Predictions using Artificial Neural Networks*, Computational Intelligence in Reliability Engineering (SCI) **40**, 197–222 (2007)
www.springerlink.com

tors. They are generally nonlinear and have the capability for generalization [12]. ANN models have recently attracted more and more attention [3, 10, 13, 16, 25, 28, 29]. ANN models are developed with respect to software failure data under specific network architecture. Compared to SRGMs, ANN models are much less restrictive in assumptions. Besides these major kinds of models, there are also some models recently developed through Fuzzy theory [2], Bayesian networks [1], etc.

Currently, most software reliability models, both analytical and data-driven models, assume (explicitly or implicitly) that software faults can be removed immediately once they are detected. As a result, these models only describe the dynamic behavior of the software fault detection process (FDP). In real practice, after a fault is detected, it has to be reported, diagnosed, removed and verified before it can be noted as corrected. This related correction time is not that trivial to be ignored [30, 39]. Furthermore, the fault-correction time is an important factor in some management decision analysis, such as stopping time for testing, fault-correction control, and fault correction resource allocation. Therefore, utilizing only software fault detection data series can result in highly inaccurate predictions of the software reliability. When both fault detection and correction data series are available, they can be utilized by incorporating fault correction process (FCP) into software reliability models to make the software reliability models more realistic.

Some extensions on current software reliability models have been explored. For analytical models, Schneidewind (1975) proposed to model fault correction process as a separate process following the fault detection process with a constant time lag [26]. This idea was extended in several ways in [35]. Schneidewind (2001) further extended the original model by assuming the time lag is a random variable [27]. These works are based on non-homogeneous Poisson process (NHPP) models where the time delay is the critical aspect of the modeling. As this approach is based on the traditional software reliability models, much time on modeling is saved. In addition, with only one extra factor of correction time, this model provides a simple analysis approach. Within the Markov framework, a non-homogeneous continuous time Markov chain has been proposed [9]. Due to its complexity, analysis is not tractable and is often done trough simulation. Moreover, there is a state explosion problem with such models. Similarly, ANN models can also be extended to model both FDP and FCP. This can be done with a separate network for FCP in addition to the original one for FDP. As a general designation, all these modeling approaches will be called separate approaches, for they are developed through a separate way.

However, applying this separate approach to either analytical or ANN model still fails to describe the interactions between FDP and FCP. Spe-

cifically, the paired analytical models treat the software FDP as a NHPP independent from FCP and no influence from FCP is considered, while the influence of FDP on FCP is described by the time delay, which is not always the case for variant software testing processes. As for a separate ANN model, no interactions between these two processes are considered at all, and the two processes are treated as uncorrelated from each other. However, the feedback of fault correction on detection can not be ignored. Intuitively, slow fault correction should have negative effects on the fault detection process, and in extreme, it can make the successive detection process halt; while fast correction process would add pressure to the fault detection indirectly, through action on the testing personnel. In addition, as a following process to FDP, FCP can be described better by incorporating more information from FDP models. None of the above described approaches can meet this requirement.

Compared with the analytical approach, the ANN modeling framework is flexible in combining multiple processes together [4], and has the potential to overcome the deficiencies described above. This problem is explored comprehensively in this chapter under this framework. The combined ANN models with both the fault detection and correction processes are studied, focusing on the incorporation of the bi-directional influences between these two processes. In comparison with the former two schemes, more accurate prediction can be expected. Specifically, comparing with paired analytical models, the combined ANN model extracts the feedback from FCP on FDP, enabling better prediction for FDP. However, as the analytical models can describe the impact of FDP on FCP with time delay assumption, these two models would compete in predicting FCP. Compared to the separate ANN model, the combined model describes better the influences between the two processes, so they can be expected to perform better in both FDP and FCP prediction. Further in this chapter two kinds of framework for the combined ANN model are proposed and comparisons among these available models are made.

This chapter is organized as follows. In section 2, an overview on traditional software reliability models and their extensions to incorporate FCP is presented. In section 3, with the formulation of this problem, combined ANN models are described in detail. Two specific frameworks are introduced, one feedforward and one recurrent, which are modeled through different approaches. Section 4 applies these two combined ANN models to real software reliability data, presenting the comparisons within the two frameworks. Detailed comparisons with the paired analytical models and separate ANN model are given in section 5. Section 6 presents our conclusions and discussions on further studies on combined ANN model.

8.2 Overview of Software Reliability Models

8.2.1 Traditional Models for Fault Detection Process

In this section, we provide an overview of the major modeling approaches, adopting the classification approach similar to [31]. Generally, software reliability growth models (SRGMs) have both analytical and data-driven models. Analytical SRGMs have three major sub-categories: non-homogeneous Poisson process (NHPP) models, Markov models and Bayesian models. They are constructed by analyzing the dynamics of the software failure process, and their applications are developed by fitting them against software failure data.

8.2.1.1 NHPP Models

Denote $N(t)$ as the cumulative number of software failures occurred by time t. The process $\{N(t); t \geq 0\}$ is assumed to follow a Poisson distribution with characteristic MVF (Mean Value Function) $m(t)$.By assuming perfect and immediate fault-correction, the failure (fault-detection) process is also a fault-removal process.

Generally, different fault detection models can be obtained by using different nondecreasing MVF $m_d(t)$. For finite $m_d(t)$ models, there are two representative models as GO-model and S-shaped NHPP model. The GO-model [8] describes the fault detection process with exponential decreasing intensity with MVF as.

$$m_d(t) = a \cdot (1 - e^{-bt}), \quad a, b > 0 \tag{2.1}$$

The S-shaped model [36] describes the fault detection process with an increasing-then-decreasing intensity, which can be interpreted as a learning process. The MVF is given as

$$m_d(t) = a \cdot [1 - (1 + bt)e^{-bt}], \quad a, b > 0. \tag{2.2}$$

In both models, a is the final number of faults that can be detected by the testing process, and b can be interpreted as the failure occurrence rate per fault.

8.2.1.2 Markov Models

The best-known software reliability model, the JM-model, is a Markov model [14]. This model has the following underlying assumptions:

- the number of initial software faults is an unknown but fixed constant;
- a detected fault is removed immediately and no new fault is introduced;
- times between failures are independent, exponentially distributed random quantities;
- all remaining software faults contribute the same amount to the software failure intensity as ψ.

Denote N_0 as the number of software faults in the software before testing starts. From the assumptions, after the kth failure, there are (N_0-k) faults left, and the failure intensity decreases to $\psi(N_0\text{-k})$. Then the time between failures T_i, i = 1, …, N_0, are independent exponentially distributed random variables with respective parameter as $\lambda(i)=\psi[N_0-(i-1)]$, i=1,…, N_0.

8.2.1.3 Bayesian Models

Bayesian analysis is a commonly accepted approach to incorporate previous knowledge in software testing. Most Bayesian formulations are based on the previous two kinds of models. One of the best-known Bayesian model is the LV-model [20]. It assumes that the time between failures are independent exponentially distributed with a parameter that is treated as random variable,

$$f(t_i \mid \lambda_i) = \lambda_i \cdot e^{\lambda_i t_i}, \; i = 1, 2, \ldots, n. \tag{2.3}$$

in which λ_i is assumed to have a Gamma prior distribution as

$$f(\lambda_i \mid \alpha, \psi(i)) = \frac{[\psi(i)]^{\alpha} \lambda_i^{\alpha-1}}{\Gamma(\alpha)} e^{\psi(i)\lambda_i} \tag{2.4}$$

where α is the shape parameter and $\psi(i)$ is the scale parameter depending on the number of detected faults.

8.2.1.4 ANN Models

ANN approach to model software is originally proposed in [15]. The reliability prediction here is regarded as an explanatory or causal forecasting problem [38]. The mapping between inputs and outputs of ANN can be written as follows: for generalization training $n_t = f(t_n)$; while for prediction

training $n_t = f(t_{n-1})$, where t is the time when n failures occur. Most of the recent models [3, 13, 25, 29] take software reliability prediction as a time series forecasting problem [38]. The mapping of ANN can be written as $y_{t+1}=f(y_t, y_{t-1},\ldots,y_{t-p})$, as illustrated in Fig. 1. Different failure data and network architectures are applied. y_t could be the inter/accumulated failure time/number. Both feedforward and recurrent neural networks have been applied. Usually, one-step predictions are developed for the measurement of failure time or number, and after that multi-step predictions can be obtained iteratively to show the trend of software failure behavior. ANN models have been successfully applied to solve software optimal release time problem with multi-step reliability prediction [7].

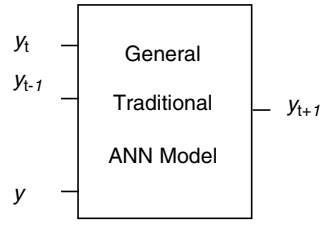

Fig. 1. General Traditional ANN Model

8.2.2 Models for Fault Detection and Correction Processes

8.2.2.1 Extensions on Analytical Models

Analytical model extensions use SRGMs to model the fault detection process, and describe the fault correction process as a time-delayed process due to time delay for correction. With FDP modeled as Schneidewind's SRGM, by assuming that fault correction has the same rate as detection, the FCP is modeled as a delayed FDP with a constant, random or time-dependent time-lag [26, 27, 35]. Extensions can be made to model FDP with other NHPP (Non-Homogeneous Poisson Process) SRGMs and the FCP can be modeled as a correspondingly delayed process. GO-Model and delayed S-shaped model are typical NHPP models, with the S-shaped model focusing on describing the learning-phenomenon along with software testing. Specifically, if FDP is modeled with GO-model, software FDP and FCP are described as two processes with the following paired characteristic MVFs (Mean Value Functions)

$$\begin{cases} m_d(t) = a(1-e^{-bt}), & a>0, b>0 \\ m_c(t) = a\left[1-e^{-b(t-\Delta_t)}\right], & t \geq \Delta_t \end{cases} \tag{2.5}$$

If FDP is modeled as delayed S-shaped model to describe the learning phenomenon, the paired MVFs are given as

$$\begin{cases} m_d(t) = a\left[1-(1+bt)e^{-bt}\right], & a>0, b>0 \\ m_c(t) = a\left[1-(1+b(t-\Delta_t))e^{-b(t-\Delta_t)}\right], & t \geq \Delta_t \end{cases} \tag{2.6}$$

where Δ_t denotes the time-delay of FCP with respect to FDP, which can be a constant or a time-dependent value.

8.2.2.2 Extensions on ANN Models

In parallel, similar to the paired analytical models describing these two processes separately, traditional ANN models can also be extended to model both FDP and FCP in a separate way. Originally, software reliability ANN models use the cumulative detected faults number data sequence $\{d_1, d_2, ..., d_n\}$ collected from FDP to establish the model presented in Fig. 2a. Separately, the FCP model can be incorporated with the cumulative corrected faults number data sequence $\{c_1, c_2, ..., c_n\}$ collected from FCP.

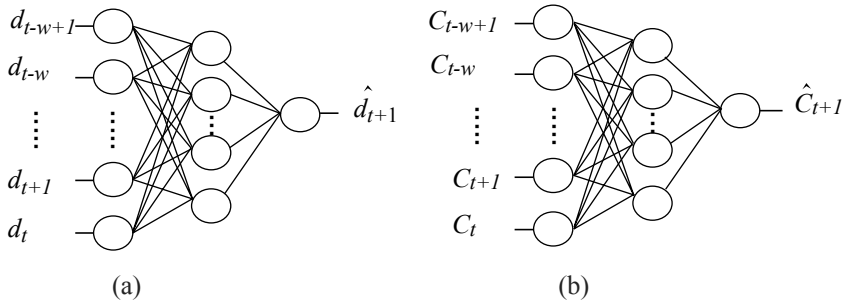

Fig. 2. Separate ANN Model Architecture, for FDP (a) and for FCP (b)

The corresponding framework is shown in Fig. 2b. As the models for FDP and FCP constitute two separate networks, they are further referred to as separate ANN models.

8.3 Combined ANN Models

In order to provide more accurate software reliability data prediction, it is essential to model the related dynamic phenomenon more realistically. Software testing (random testing) is a complicated and interactive process, and from the viewpoint of software reliability, there are both software fault detection and fault correction processes. These two processes are correlated. Once a fault is detected, it will be submitted for correction. This requires time for diagnosing, removal and verification. If the fault does not hamper the detection process, these two processes will proceed in parallel, but if it is so severe that the software is deemed inoperable, the detection process would wait until the fault is corrected; if the detection rate is very high, it will bring pressure to correction process, and vice versa.

Traditional SRGMs and ANN models only describe the fault detection process by assuming immediate and perfect correction. The practical extensions, paired analytical models and separate ANN models mentioned in the previous section, account for the fault correction process. However, they fail to model the interactions between these two processes. In this section, we propose the combined ANN models, as illustrated in Fig. 3, to model both FDP and FCP using the method of multivariate time series prediction [4] and to incorporate the interactions between these two processes.

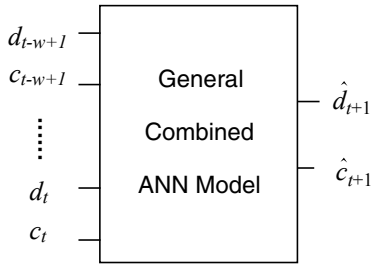

Fig. 3. General combined ANN Model

The architecture combines the two processes in both the input and output of the ANN model, and is the reason they are called combined ANN models.

Specifically, there are two major kinds of ANNs: feedforward and recurrent, and both have their advantages in time series predictions. Feedforward ANNs have been adopted by most researchers and there are some "rules of thumb" to follow in modeling network architecture. Construction with this well-studied framework is effortless and effective. Although recurrent ANNs are less studied, they have the ability to incorporate tempo-

ral information as they feedback inner states or outputs into the input layer. Both frameworks will be explored in this chapter.

8.3.1 Problem Formulation

Within the framework of neural networks modeling [4, 15], we formulate our problem as follows. By denoting $D(i), i = 1, 2, 3...$ and $C(i), i = 1, 2, 3...$, as the cumulative number of detected faults and corrected faults after testing period i respectively, we define software testing process as a bi-process combining both FDP and FCP, $S(i) = \begin{bmatrix} D(i) \\ C(i) \end{bmatrix}, i = 1, 2, 3....$ With ongoing testing process, software fault-related data can be collected as data sets of $\{s_1, s_2, ..., s_{t-1}, s_t\} = \{\begin{bmatrix} d_1 \\ c_1 \end{bmatrix}, \begin{bmatrix} d_2 \\ c_2 \end{bmatrix}, ..., \begin{bmatrix} d_{t-1} \\ c_{t-1} \end{bmatrix}, \begin{bmatrix} d_t \\ c_t \end{bmatrix}\}$, from the beginning of software system random testing until current testing period t. $D(i)$ and $C(i), i = 1, 2, 3...$ are two correlated processes. To make testing related decisions, at the end of every testing period, we are interested in knowing the possible outcomes of the following time period. In other words, we need to develop one-step predictions based on the historical data sequence $\{s_1, s_2, ..., s_{t-1}, s_t\}$ to get $\hat{S}_{t+1} = \begin{bmatrix} \hat{D}_{t+1} \\ \hat{C}_{t+1} \end{bmatrix}$, the predicted number of cumulative faults by the end of testing period t+1. Then with the updating of new data s_{t+1}, we can evaluate the performance of the previous prediction and develop the prediction for the fault number in the next interval $\hat{S}_{t+2} = \begin{bmatrix} \hat{D}_{t+2} \\ \hat{C}_{t+2} \end{bmatrix}$. The prediction process is continually updated as new testing data becomes available from ongoing testing.

8.3.2 General Prediction Procedure

With pre-set configurations of the network, the prediction is a sequentially updating process, with stepwise prediction utilizing each newly collected software faults data from the ongoing testing. At each point, with the latest and all past data, the network is retrained for new prediction in three

specific steps: data normalization, network training, and prediction. The specific prediction procedure for any one point with the combined ANN model is described generally as follows.

8.3.2.1 Data Normalization

Collected cumulative software fault data $\{s_1,s_2,...,s_{t-1},s_t\}$ cannot be fed into networks directly as they need to be normalized between [0, 1]. Normalization functions varies, and for our case, the simple normalization scheme of $s_i^{norm}=s_i/s_{max}$ is adopted. As $D(i)$ and $C(i)$, i = 1,2,3… are incremental processes, the collected or predicted data would show an increasing trend, so we need to estimate the upper limit of $\hat{D}(t+1)$ and $\hat{C}(t+1)$. With the available cumulative data, this value can be calculated by estimating the maximum possible increments of Δd and Δc. This number can be estimated from past experience of similar projects. Then $s_{\max}$ at the end of testing period t is calculated as

$$s_{\max}^{t} = Max(d_t, c_t) + Max(\Delta d, \Delta c) \tag{3.1}$$

To simplify our notation further we assume that $\{s_1, s_2, \ldots, s_{t-1}, s_t\}$ is already normalized.

8.3.2.2 Network Training

With available normalized data, the neural networks with pre-defined configuration can be trained to model these two processes. The collected historic data sequence $\{s_1,s_2,...,s_{t-1},s_t\}$ should be grouped into as many as t-w past-to-future mapping patterns denoted as $\{s_{k-w,}s_{k-w+1},\ldots,s_{k-1} \mid s_k\}, k = w+1,\ldots,t$ .These training patterns abstract the historic input-output relationships of the network. The patterns are used to train the network by adjusting its weights and bias, which are initially set randomly. Typically, backpropagation algorithms are used to train the networks and there are some variations in the algorithms. These algorithms usually look for ANN parameters (weights of internodes' connections and node biases) to fit the patterns by minimizing the deviation of the network outputs from the outputs of training patterns. To overcome the overfitting problem, usually the generalization technique is adopted.

8.3.2.3 Fault Prediction

With the trained network, which has "fit" the training patterns out of the collected data set $\{s_1, s_2, ..., s_{t-1}, s_t\}$, we can use the most recent *w* data set $\{s_{t-w+1}, s_{t-w+2}, \ldots, s_t\}$ to generate the next pattern as $\{s_{t-w+1}, s_{t-w+2}, \ldots, s_t \mid \hat{s}_{t+1}\}$. Then we can get our predictions for the next time point as $\hat{s}_{t+1} = \begin{bmatrix} \hat{d}_{t+1} \\ \hat{c}_{t+1} \end{bmatrix}$.

An initialization problem exists in the training algorithms. Different initial values for network weights and bias would generate different training results. For the generalized training algorithm adopted here, the initial values are assigned randomly. For each point predictions, *m* replicated runs are usually performed with different initializations, and the mean is used as the prediction outputs [15, 29] given as

$$\hat{s}_{t+1} = \frac{1}{m} \sum_{i=1}^{m} \hat{s}_{t+1}^{i} \tag{3.2}$$

8.3.3 Combined Feedforward ANN Model

8.3.3.1 ANN Framework

The framework of the combined feedforward ANN model is illustrated in Fig. 4.

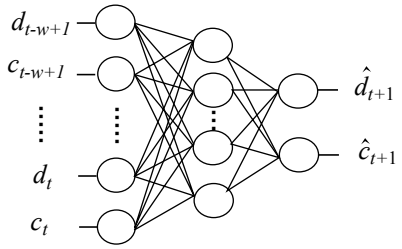

Fig. 4. Combined Feedforward ANN Model Architecture

It has inputs of both $\{d_{t-w+1}, .., d_t\}$ and $\{c_{t-w+1}, .., c_t\}$ and outputs of both $\hat{d}_{t+1}$ and $\hat{c}_{t+1}$. Specifically, the model is trained with the data from the bi-process $\{s_1, s_2, ..., s_{t-1}, s_t\}$ (both $\{d_{t-w+1}, .., d_t\}$ and $\{c_{t-w+1}, .., c_t\}$),

combined with the past information of these two interactive processes together. Then with the well trained networks, the prediction can be generated from the latest w data points $S_{t-w+1},..,S_{t-1},S_t, 0 \le w \le t$ as the following function:

$$\hat{S}_{t+1} = F(S_{t-w+1},\ldots,S_{t-1},S_t\} \tag{3.3}$$

8.3.3.2 Performance Evaluation

As an on-line prediction procedure, it starts tracking from the early stages of software testing. With the ongoing testing process, prediction is developed with the arrival of every updated data. For each single point prediction, the prediction is expected to be close to the collected data in the coming time period. Therefore, the prediction can not be evaluated until the next updated data is collected. As a whole, the prediction performance of the ANN model is evaluated with respect to all the past predictions with the data obtained from the whole testing process.

Specifically, suppose dataset $\{s_{1,}s_2,\ldots,s_t\}$ is used for network configuration. Within this data set, we simulate the sequential stepwise prediction process as in real software testing. Assume t_0 is the first point for prediction, and all the preceding data points $\{s_{1,}s_2,\ldots,s_{t_0-1}\}$ are used to train the network to get the prediction $\hat{s}_{t_0}$. With m different network initialization, m prediction repetitions are developed as $\hat{s}_{t_0,j}$, $j = 1,2,\ldots,m$. This procedure is carried on to get the following stepwise predictions $\hat{s}_{i,j}$, $i = t_0, \ldots, t$, $j = 1,2,\ldots,m$.

The prediction of each point is the average of the m repetitions $\hat{s}_i = \frac{1}{m}\sum_{j=1}^{m}\hat{s}_{i,j}$, $i = t_0, \ldots, t$ and the performance is evaluated with its departure from the actual data as

$$SE_i = \left|\hat{s}_i - s_i\right|^2 = (\hat{d}_i - d_i)^2 + (\hat{c}_i - c_i)^2 = SE_i^d + SE_i^c,\ i = t_0, \ldots, t.$$

It is expected that the selected model works well through the whole testing process. The overall performance of the configuration is then determined by

$$MSE = \frac{1}{t - t_0 + 1} \sum_{i=t_0}^{t} SE_i^d + \frac{1}{t - t_0 + 1} \sum_{i=t_0}^{t} SE_i^c = MSE^d + MSE^c \quad (3.4)$$

8.3.3.3 Network Configuration

Feedforward networks are the most common networks and are widely studied. There are several "rules of thumb" to develop these networks that we adopt here. Within the context of this specific feedforward model, we pre-configure the network as follows. The architecture has three layers: an input layer, a hidden layer, and an output layer. Each ANN has 2**w* inputs, which corresponds to *w* data sets [d, c]' presented to the network. In order to overcome the overfitting problem, the number of the hidden nodes should not be large. By comparing some practical recommendations, we chose this number as double the number of input nodes [38], i.e., 4**w*. The sigmoid function (logistic) is used as the activation function for each node in both the hidden and the output layers.

Using Eq. 3.4 as the performance criterion, the trial and error approach is used to determine the remaining parameters of the training algorithm.

8.3.4 Combined Recurrent ANN Model

8.3.4.1 ANN Framework

Similar to the combined feedforward ANN model, the proposed recurrent ANN model has the combined architecture as shown in Fig. 5, with feedback from the inner states to the input layer. Similarly, the model is trained with the data from the bi-process $\{s_1, s_2, ..., s_{t-1}, s_t\}$ (both $\{d_{t-w+1}, .., d_t\}$ and $\{c_{t-w+1}, .., c_t\}$), combined with the past information of these two interactive processes together. Then with the well trained networks, the prediction can be generated from the latest *w* data points $S_{t-w+1}, .., S_{t-1}, S_t, 0 \le w \le t$ as the following function:

$$\hat{S}_{t+1} = F(S_{t-w+1}, \ldots, S_{t-1}, S_t; State_t\} \quad (3.5)$$

With respect to the model constraints, some parameters can be pre-configured as follows. Similar to the feedforward architecture, the basic Elman adopted here has architecture of three layers, one input layer, one

hidden layer, and one output layer. Differently, Elman network has feedback from the hidden states into the network inputs. This network has $2*w$ inputs, which corresponds to w data sets [d, c]' presented to the network. The sigmoid function (logistic) is used as the activation function for each node in both the hidden and the output layers.

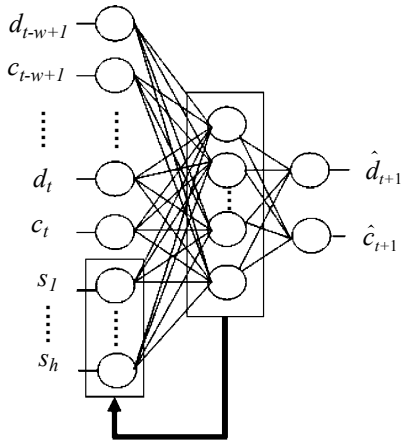

Fig. 5. Combined Recurrent ANN Model Architecture

8.3.4.2 Robust Configuration Evaluation

Unlike the evaluation on the combined feedforward network, the performance for the combined recurrent network is evaluated differently. Similarly, for each single point prediction, the prediction is expected to be close to the collected data in the coming time period. In addition, because the prediction is random, some repetitions are generated and small variance is also expected. With a given dataset, which represents the history of a period of software testing, the configuration of network can be evaluated with its average performance in prediction through this period from the first prediction point. Different from the evaluation for the combined feedforward ANN model, robustness criterion is adopted for combined recurrent model.

The prediction of each point is the average of the m repetitions $\hat{s}_i = \frac{1}{m}\sum_{j=1}^{m} \hat{s}_{i,j}$, $i = t_0, \ldots, t$ and the performance is evaluated by its departure from the actual data

$$SE_i = \left|\hat{s}_i - s_i\right|^2 = (\hat{d}_i - d_i)^2 + (\hat{c}_i - c_i)^2 = SE_i^d + SE_i^c, \; i = t_0, \ldots, t$$

The dispersion of these m repetitions is given as

$$S_i^2 = \frac{1}{m}\sum_{j=1}^{m}(\hat{d}_{i,j} - \hat{d}_i)^2 + \frac{1}{m}\sum_{j=1}^{m}(\hat{c}_{i,j} - \hat{c}_i)^2 = S_{i,d}^2 + S_{i,c}^2,\ i = t_0, \ldots, t.$$

It is expected that the selected model works well through the whole testing process, and the overall performance of the configuration is evaluated by the following two criteria

$$\begin{cases} MSE = \dfrac{1}{t - t_0 + 1}\sum_{i=t_0}^{t} SE_i = MSE^d + MSE^c \\ MS^2 = \dfrac{1}{t - t_0 + 1}\sum_{i=t_0}^{t} S^2 = MS_d^2 + MS_c^2 \end{cases} \tag{3.6}$$

However, as both criteria can be contradictory, in to balance both the prediction location and dispersion, their summation is used to evaluate the performance instead. Hence, for one specific network configuration θ, the performance function is expressed as

$$L(\theta) = MSE + MS^2 \tag{3.7}$$

8.3.4.3 Network Configuration through Evolution

There is less guidance for recurrent network configuration, and also the network training requires much more time than feedforward networks. Therefore some automatic "trial-and-error" approach is useful. Evolutionary programming provides an approach to optimize complex problems with specific fitness function, which suits our problem well, i.e., to search for an optimal configuration setting θ^* from the parameter space with respect to fitness function of $L(\theta)$ in Eq. 3.7. In fault detection prediction, genetic algorithm has been applied to optimize the network architecture parameters to determine the number of inputs and hidden nodes for feedforward architecture [29]. Specially, we also make the configuration setting include the algorithm parameters, for they are found to have great influence on the performance.

The configuration evolving process is described as following steps:

- *Step 1:* Encode the configuration setting θ into chromosome.
- *Step 2:* Generate an initial population of l individuals $\theta_1, \theta_2, ..., \theta_l$
- *Step 3:* Calculate the fitness function $L(\theta_i)$, $i=1,2,\ldots,l$ for each individual

– *Step 4:* Select parent settings for next generation according to the fitness values:
  - Crossover breeding operator
  - Mutation operator
  - Cull inferior solution
– *Step 5.* Repeat step 3 until stopping criteria are met and return optimal setting θ^*.

8.4 Numerical Analysis

To illustrate the application of combined ANN models we apply the two suggested models to real data collected from a middle sized application software testing process. The collected interval data set includes both fault detection and correction data, $\Delta D(t)$ and $\Delta C(t)$, as shown in Table 1.

Table 1. Fault detection and correction data (number per week)

Week	$\Delta d(t)$	$d(t)$	$\Delta c(t)$	$c(t)$
1	12	12	3	3
2	11	23	0	3
3	20	43	9	12
4	21	64	20	32
5	20	84	21	53
6	13	97	25	78
7	12	109	11	89
8	2	111	9	98
9	1	112	9	107
10	2	114	2	109
11	2	116	4	113
12	7	123	7	120
13	3	126	5	125
14	2	128	2	127
15	4	132	0	127
16	9	141	8	135
17	3	144	8	143

The proposed combined ANN models are used to develop one-step prediction for both $D(t)$ and $C(t)$, starting from some early point and tracking the software testing process till the end with the continuous updating of collected cumulative data .

With the combined ANN model, using either the feedforward or recurrent network architecture, some pre-configuration can be developed with

respect to the constraints of the specific problem. At the beginning of this testing process, available data is scarce. Therefore, prediction needs to be developed as soon as possible, providing timely decision-making assistance for the testing procedure. However, for such data-driven modeling approaches like ANN, the model cannot be well adjusted without essential number of data points. As a result prediction cannot begin until enough data is collected. To compromise, the size of sliding window w cannot be large, and we set it to $w = 3$, and start prediction from the 6th testing period. Then we know our combined ANN model will have 6 inputs. As to the number of hidden nodes and some parameters related to algorithm, they are configured differently for feedforward and recurrent networks. Obviously, the network model has two outputs.

8.4.1 Feedforward ANN Application

As a common "rule of thumb", the number of hidden nodes is set to be double the number of input nodes. With different configurations on the training algorithm parameters, the following procedure is developed with trial-and-error to get a fully-configured network for further prediction out of sample. With the available data sequence as $\{s_1, s_2, ..., s_{t-1}, s_t\}$ the prediction can be developed as follows.

1. Data normalization:

Based on experience from similar past projects and current testing personnel allocation, the expected incremental number of $\Delta D(t)$ and $\Delta C(t)$ cannot exceed 25: $Max(\Delta d, \Delta c) = 25$. This number if set fixed for the whole prediction process with ongoing testing process. The data is normalized with the maximum number calculated from Eq. 3.1.

2. Network training:

With the normalized data set, the training patterns are generated for both frameworks respectively as $\{s_{k-w}, s_{k-w+1}, \ldots, s_{k-1} \mid s_k\}, k = w+1, \ldots, t$.

For our case, backpropagation algorithm is adopted to train the ANN with the generated patterns. To improve generalization of the training, the regularization method is implemented by adding the mean of the sum of squares of network weights and biases g_i, $i = 1,\ldots,l$, MSW, to the network performance function MSE in the following form

$$MSE_{reg} = \gamma \cdot MSE + (1-\gamma) \cdot MSW \tag{4.1}$$

where γ is the performance ratio, and $MSW = \frac{1}{l}\sum_{j=1}^{l} g_j^2$. Such regularization can force the network to have smaller weights and biases, which provides the smoother network response. It also reduces the chance of overfitting. The parameter γ is set by trial and error

3. Prediction:

With the well-trained network, the latest *w* data is fed and the prediction is generated as the network outputs. 50 runs of prediction for each point are performed, yielding 50 predictions for point *i* , $\hat{s}_{i,j}$, *i = 6, j = 1, …, 50*. Related variances are calculated to estimate the robustness of the model.

The prediction process is performed from the 6th testing period till the end of the testing. The prediction for each point is evaluated with the updated fault data. The prediction sequence is obtained in the form $\left\{\hat{d}_6, \hat{d}_7, ..., \hat{d}_{16}, \hat{d}_{17}\right\}$ and $\left\{\hat{c}_6, \hat{c}_7, ..., \hat{c}_{16}, \hat{c}_{17}\right\}$. Then the prediction performance of the model over the whole testing process is evaluated by comparing with the true data with mean squared errors calculated with Eq. 3.4. The corresponding prediction results are summarized in Table 2.

Table 2. One-step Predictions with Combined Feedforward ANN Model

Week	$\hat{d}(t)$		SE_t^d	$\hat{c}(t)$		SE_t^c
	Mean	Var		Mean	Var	
6	96.29	0.4324	0.50	73.43	0.7162	20.87
7	102.48	0.0895	42.55	94.53	0.9200	30.58
8	113.21	0.0032	4.88	91.66	0.0138	40.23
9	116.02	0.0024	16.12	97.76	1.0854	85.47
10	112.80	0.0838	1.43	110.17	0.9192	1.38
11	113.38	0.0550	6.86	111.63	0.0029	1.87
12	118.28	0.0024	22.31	115.25	0.0012	22.53
13	129.78	0.0062	14.30	123.50	0.0097	2.25
14	126.86	0.0682	1.29	127.18	0.0906	0.03
15	128.68	0.1775	11.03	129.45	0.0036	5.99
16	134.82	0.0019	38.21	130.10	0.0153	23.99
17	144.54	0.0066	0.29	140.42	0.0011	6.66
Ave.		0.0774	13.31		0.3149	20.15

Var in the table denotes the variance of the repeated predictions at each point. From Table 2, we can see that under this network configuration, the predictions along the period of this dataset can fit the observed value well with small variances.

8.4.2 Recurrent ANN Application

Adopting similar preset architecture parameters as the feedforward model, the remaining architecture parameter for the recurrent model is the number of hidden nodes n_h. Besides the architecture parameter, the network configuration θ should also include critical training algorithm parameters. The specific prediction procedure for this dataset is similar to that in section 3.1. From the former analysis, we have found that the performance ratio γ is a critical parameter. Here, back-propagation algorithm with learning rate and momentum is adopted. These two parameters are important to the algorithm performance. As the learning rate is adaptive, it is important for network training to set a proper value for momentum m_o. The network parameters to be configured can be determined as $\theta = [n_h \quad m_o \quad \gamma]$, i.e. the hidden nodes number, the momentum, and the performance ratio. For each specific configuration, such a prediction process is performed from the 6th testing period till the last one, obtaining predictions $\hat{s}_{i,j}$, i = 6,7,…,17, j =1,2,…,5. The corresponding fitness function value, i.e., the network performance value, can be calculated through

$$L(n_h, m_o, \gamma) = MSE + MS^2 .$$

This way, the evolving procedure in the former section is developed to find the proper value of $\theta^* = [n_h^* \quad m_o^* \quad \gamma^*]$.

Table 3. One-step Predictions with Combined Recurrent ANN Model

Week	$\hat{d}(t)$		SE_t^d	$\hat{c}(t)$		SE_t^c
	Mean	Var		Mean	Var	
6	95.68	0.3919	1.76	72.64	1.4984	28.71
7	102.60	0.5183	40.96	86.78	1.9346	4.94
8	112.93	0.1001	3.72	98.83	0.2321	0.69
9	114.82	0.0887	7.95	102.81	0.1440	17.56
10	116.62	0.0113	6.86	108.82	0.0476	0.03
11	117.85	0.0376	3.42	110.86	0.1338	4.58
12	119.31	0.0517	13.62	113.52	0.1996	41.99
13	124.07	0.1142	3.72	121.06	0.3803	15.52
14	127.34	0.0099	0.44	126.34	0.0332	0.44
15	129.51	0.0708	6.20	129.85	0.1820	8.12
16	131.65	0.0000	87.42	132.73	0.0000	5.15
17	136.02	0.0389	63.68	137.77	0.0815	27.35
Ave.		0.1194	19.98		0.4056	12.92

With respect to this dataset, a configuration of network has been evolved with genetic algorithm as the hidden nodes number = 14; performance ratio = 0.9450; momentum = 0.9711. With this configuration, 20 more repeated predictions are obtained again for each time point. The prediction results are shown in the Table 3.

From these results, we can see that under this network configuration, the predictions along the period of this dataset can fit the observed value very well with small variances.

8.4.3 Comparison of Combined Feedforward & Recurrent Model

Both these two types of ANN models have been applied to model software reliability prediction. These two architectures have been also compared through different criteria with respect to different dataset [7, 13, 15, 29]. Although Elman architecture is advocated to incorporate the temporal patterns, there is no consistent advantage from these experimental results. As far as our dataset is concerned, we compare these two architectures with their predictive performance using both location (MSE) and dispersion (MS^2). This is summarized in Table 4.

Table 4. Comparison: Combined Feedforward *VS* Recurrent ANN

	MSE	MS2	L
Combined Feedforward ANN models	33.46	0.3923	33.8523
Combined Recurrent ANN models	32.90	0.5250	33.4250

From this table, we can see that there is slight advantage of combined recurrent ANN model over feedforward model, with respect to the "robust" performance L. However, if we take the criteria of either *MSE* or MS^2, contradictory conclusions will be drawn, although the differences are small. With respect to this data set, these two models are nearly the same. Therefore, both configured models can be set to develop predictions for the coming data points. Comparatively, combined feedforward ANN model would be more effective.

8.5 Comparisons with Separate Models

In this section, we proceed to verify that the proposed combined model would perform better than separate models. Accordingly, the comparisons

of the combined ANN models with those two separate models mentioned in section 1 are developed.

8.5.1 Combined ANN Models vs Separate ANN Model

In the separate ANN model (Fig. 1.) two separate networks are modeled: one for $D(t)$ and the other for $C(t)$. The comparison between combined and separate ANN models is of interest because both of them are data-driven ANN models and their differences would focus on the effectiveness of the incorporating the correlations between these two processes. For the data set in Table 1, the "online" prediction process is developed with separate ANN model as follows. Feedforward network is adopted and the configuration is as follows. In the combined ANN models, the size of sliding window is set at w=3 and the prediction starts from the 6th point. The training, prediction and evaluation procedures are also the same as combined models. The prediction results are listed in Table 5.

Table 5. One-step Predictions with Separate ANN Model

Week	$\hat{d}(t)$		SE_t^d	$\hat{c}(t)$		SE_t^c
	Mean	Var		Mean	Var	
6	95.64	0.2296	1.84	66.17	0.3204	139.85
7	102.35	0.0263	44.23	89.78	1.4164	0.61
8	114.27	3.4484	10.69	91.49	0.0002	42.33
9	116.49	0.0007	20.18	100.13	0.0306	47.22
10	108.13	0.0039	34.41	109.78	0.0025	0.61
11	113.69	0.0014	5.35	112.23	0.0003	0.59
12	114.90	0.0000	65.68	114.89	0.0001	26.09
13	122.67	5.6761	11.12	117.44	0.0002	57.08
14	129.32	0.9281	1.75	125.94	0.0002	1.13
15	130.79	0.3720	1.47	129.84	0.0010	8.08
16	134.54	0.0515	41.71	129.80	0.0006	27.01
17	144.07	0.0002	0.00	135.14	0.0004	61.85
Ave.		0.8949	19.87		0.1477	34.37

From the results shown in Tables 2 - 5, we can compare these two kinds of models in two ways. With respect to the overall performance of MSE, the combined models outperform the separate one, which verifies the advantages of modeling the two processes together. In addition, from the prediction performance for each point, SE_t^d and SE_t^c, we observe an interesting phenomenon: prediction of the first point for $\hat{C}(t)$ is not accept-

able, however the first prediction for $\hat{D}(t)$ performs well. This reflects the delay of FCP over FDP, which results in some data shortage for prediction of $\hat{C}(t)$ at the initial phase. Fortunately, prediction is reinforced by combining outputs of $C(t)$ and $D(t)$ in the combined ANN model.

8.5.2 Combined ANN Models vs Paired Analytical Model

When applying paired analytical models to the fault data, one faces the problem of model selection for FDP from many available NHPP SRGMs. As far as our case (Table 1) is concerned, the interval detected faults show an increasing trend in the early phase of software testing. Delayed S-shaped NHPP model is designed to describe such learning phenomenon. In addition, as this project takes relatively short testing period and is common application software, detected faults should be common ones and they are handed to available correctors that are stable though testing process. Therefore, instead of using Schneidewind's model directly, the slight extension as described in Eq. 2.6 is adopted, assuming constant time-delay between FDP and FCP.

In a similar way, "on-line" prediction is developed by fitting the model against historical data collected with the ongoing testing process. As a model-driven method, the prediction can be started from earlier points. However, in order to compare with the combined ANN model in the same time horizon, the prediction is also developed from the 6^{th} point. The application results of the actual data with analytical models are presented in Table 6.

From the results shown in Tables 2, 3 and 6, we can see that combined ANN model performs over analytical model in prediction of both fault detection and correction. Further observations show that large prediction errors happen in the 8^{th}, 9^{th}, 16^{th} and 17^{th} points. Referring to Table 1, we see that these are the points where some unusual changes happen. Comparatively, ANN models work better on these points, showing more flexibility and sensitivity to the abnormal change. This difference can also be regarded as the difference in prediction approaches. The analytical model develops the prediction through fitting the historical data with respect to time; however, the ANN models develop networks to fit input-output patterns which incorporate the trend of data inside. More importantly, the simple time-delay assumption between the relationship of fault detection and correction does not fit this dataset well. The ANN models perform better in capturing the correlated relationships between these two processes.

Table 6. One-step Prediction with Paired Analytical Models

Week	$\hat{d}(t)$	SE_t^d	$\hat{c}(t)$	SE_t^c
6	100.97	15.74	72.71	27.95
7	113.36	18.99	91.17	4.71
8	120.71	94.28	103.71	32.64
9	121.48	89.79	110.07	9.45
10	121.55	57.00	114.10	26.05
11	120.97	24.72	116.19	10.18
12	120.93	4.29	117.80	4.82
13	122.80	10.22	120.56	19.73
14	124.78	10.36	123.13	14.98
15	126.34	32.04	125.13	3.49
16	127.70	176.84	126.81	67.01
17	130.37	185.82	129.65	178.27
Ave.		**55.15**		**28.95**

As a short conclusion, the software fault detection and correction processes are two correlated processes, and to develop accurate predictions, information about both of them should be incorporated into the model. Combined ANN models are a flexible way to implement this. Paired analytical model can describe one-directional effects, and in some cases it can perform better. However, the combined ANN models provide a unified approach to model the two processes together, which is more favorable than the analytical approach since more effort is needed on model selection in the analytical approach.

8.6 Conclusions and Discussions

In this chapter we have studied the use of neural networks to model both the software fault detection and correction processes together (referred to as combined ANN model), focusing on describing the interactions between these two correlated processes. This approach is regarded as an extension of separate ANN model under the same modeling framework, and is a complement to analytical models which only describe the influence of FDP on FCP as a time delay. With practical software testing data, this approach shows its advantage in incorporating more information than the separate ANN model and paired analytical model. Also, within the combined ANN models, both feedforward and recurrent frameworks perform well with the given dataset.

The combined ANN models are beneficial in incorporating the correlation between FDP and FCP. They model the software debugging process more realistically, with more accurate predictions. However, this model

still has some aspects for further investigation. First, faults number is one important measure of software reliability, and predictions on some other measure such as detection rate would be interesting. [33] showed detection rate can be assumed to be the same as earlier projects/versions, and ANN models would help abstract this information when datasets from previous projects are available. Second, FCP is different from FDP, where some fault-correction factors (such as personnel) can be controlled. With more understanding of the interactions between FDP and FCP, some useful software fault correction policies can be proposed for more effective testing resource allocation. Third, software reliability prediction is just an initial step of reliability analysis. The prediction results need to provide assistance on decision-making for testing management. With potential to provide more accurate and multi-step predictions, and with the modeling of both fault-detection and correction processes, the combined ANN models are expected to be more helpful in testing management, such as decisions on stopping time. Application of this software reliability approach to decision problems with larger datasets will be useful in further understanding its potential.

Acknowledgement

This research is partly supported by the National University of Singapore under the research grant R-266-000-020-112, “Modelling and Analysis of Firmware Reliability”.

References

[1] Bai CG, Hu QP, Xie M (2005) Software failure prediction based on a Markov Bayesian network model. Journal of Systems and Software 74(3) pp 275-282

[2] Cai KY (1996) Fuzzy Methods in Software Reliability Modeling. In: Introduction to Fuzzy Reliability. Kluwer Academic Publishers, pp. 243–276

[3] Cai KY, Cai L, Wang WD, Yu ZY, Zhang D (2001) On the neural network approach in software reliability modeling. Journal of Systems and Software 58(1) pp. 47-62

[4] Chakraborty K, Mehrotra K, Mohan CK, Ranka S (1992) Forecasting the Behavior of Multivariate Time-Series Using Neural Networks. Neural Networks 5(6) pp. 961-970

[5] Crow L H, Singpurwalla N D (1984) An empirically developed Fourier series model for describing software failures. IEEE Transactions on Reliability 33 pp. 176-183

[6] Dai YS, Xie M, Poh KL, Yang B (2003) Optimal testing-resource allocation with genetic algorithm for modular software systems. Journal of Systems and Software 66(1) pp. 47-55
[7] Dohi T, Nishio Y, Osaki S (1999) Optimal software release scheduling based on artificial neural networks. Annals of Software Engineering 8 pp. 167-185
[8] Goel AL, Okumoto K (1979) Time dependent error detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability 28(3) pp. 206-211
[9] Gokhale SS, Lyu MR, Trivedi KS (2004) Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain. Software Quality Journal 12(3) pp. 211-230
[10]Guo P, Lyu MR (2004) A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. Neurocomputing 56 pp. 101-121
[11]Han CC, Shin KG, Wu J (2003) A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. IEEE Transactions on Computers. 52(3) pp. 362-372
[12]Haykin S (1999) Neural networks: A comprehensive foundation. New York: Macmillan College Publishing Company
[13]Ho SL, Xie M, Goh TN (2003) A study of the connectionist models for software reliability prediction. Computers & Mathematics with Applications 46(7) pp. 1037-1045
[14]Jelinski Z, Moranda PB (1972) Software reliability research. In: Freiberger, W. (eds) Statistical Computer Performance Evaluation, Academic Press, New York
[15]Karunanithi N, Whitley D, Malaiya YK (1992) Prediction of software reliability using connectionist models. IEEE Transactions on Software Engineering 18(7) pp. 563-574
[16]Khoshgoftaar T M, Szabo RM (1996) Using neural networks to predict software faults during testing, IEEE Transactions on Reliability 45(3) pp. 456-462
[17]Kimura M, Toyota T, Yamada S (1999) Economic analysis of software release problems with warranty cost and reliability requirement. Reliability Engineering and System Safety 66(1) pp. 49-55.
[18]Kuo SY, Huang CY, Lyu MR (2001) Framework for modeling software reliability, using various testing-efforts and fault-detection rates. IEEE Transactions on Reliability. 50(3) pp. 310-320.
[19]Levitin G (2005) Reliability and performance analysis of hardware-software systems with fault-tolerant software components. Reliability Engineering & System Safety In Press, Corrected Proof, Available online 27 June 2005.
[20]Littlewood B, Verral J L (1973) A Bayesian reliabiltiy growth model for computer software. Applied Statistics 22 pp. 332-346.
[21]Littlewood B, Wright D, (1997) Some conservative stopping rules for the operational testing of safety critical software. IEEE Transactions on Software Engineering 23(11) pp. 673-683.
[22]Lyu MR (1996) Handbook of software reliability engineering. IEEE Computer Society Press

[23]Musa JD, Iannino A, Okumoto K (1987) Software reliability: measurement, prediction, application. McGraw-Hill, New York.
[24]Pham H (2000) Software reliability. Springer, Singapore; New York
[25]Sitte R (1999) Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. IEEE Transactions on Reliability 48(3) pp. 285-291
[26]Schneidewind NF (1975) Analysis of error processes in computer software. Proceedings of International Conference on Reliable Software, 1975, IEEE Computer Society, pp. 337-346
[27]Schneidewind NF (2001) Modelling the fault correction process. Proceedings of the 12th International Symposium on Software Reliability Engineering, 2001, pp.185-190
[28]Takada Y, Matsumoto K, Torii K (1994) A Software-Reliability Prediction Model Using a Neural-Network. Systems and Computers in Japan 25(14) pp. 22-31
[29]Tian L, Noore A (2005) Evolutionary neural network modeling for software cumulative failure time prediction. Reliability Engineering & System Safety 87(1) pp. 45-51
[30]Tian L, Noore A (2005) Modeling distributed software defect removal effectiveness in the presence of code churn. Mathematical and Computer Modelling. 41(4-5) pp. 379-389.
[31]Xie M (1991) Software reliability modelling, World Scientific, Singapore.
[32]Xie M, Hong GY (1999) Software release time determination based on unbounded NHPP model. Computers & Industrial Engineering. 37(1-2) pp. 165-168
[33]Xie M, Hong GY, Wohlin C (1999) Software reliability prediction incorporating information from a similar project. Journal of Systems and Software. 49(1) pp. 43-48.
[34]Xie M, Yang B (2003) A study of the effect of imperfect debugging on software development cost. IEEE Transactions on Software Engineering 29(5) pp. 471-473
[35]Xie M, Zhao M (1992) The Schneidewind software reliability model revisited. Proceedings of 3rd International Symposium on Software Reliability Engineering, 1992, pp. 184-192
[36]Yamada S, Ohba M, Osaki S (1983) S-shaped reliaibility growth modeling for software error detection. IEEE Transactions on Reliability. 32(4) pp. 289-292
[37]Yamada S, Ichimori T, Nishiwaki M (1995) Optimal allocation policies for testing-resource based on a software reliability growth model. Mathematical and Computer Modelling. 22(10-12) pp. 295-301
[38]Zhang GQ, Patuwo BE, Hu MY (1998) Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting. 4(1) pp. 35-62
[39]Zhang XM, Pham H (2005) Software field failure rate prediction before software deployment. Journal of Systems and Software. In Press, Corrected Proof, Available online 12 July 2005.