International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-4 Number-1 Issue-14 March-2014

Faults Prediction for Component Based Software using Interrelated Feed Forward Neural Networks

Deepak Shudhalwar[1], P. K. Butey[2]

Abstract

The software reliability generally depends upon the rate of failures, or the number of faults occurred in mean execution time and the numbers of faults are expected to occur during the course of execution of software. There are various models have been proposed to predict the expected number of faults for various types of software structure. In this paper we are predicting the number of expected faults occurred in each component of software as well as for the whole software in the expected execution time interval using component based neural network architecture. The simulation design and implementation results are suggesting there may be more number of predicted faults in components than the number of predicted faults in the complete software.

Keywords

Feed forward neural networks, fault Predication, Component based software architecture, Software reliability.

1. Introduction

Software Reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1]. IEEE defines the reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time [2]." Software reliability is the probability of a software system to perform its specified functions correctly over a long period of time or for different input set under the usage environments similar to that of its target customer [3]. The Software reliability is defined in terms of expected number of faults occurs in mean execution time interval.

Manuscript received March 18, 2014.

Deepak Shudhalwar, Research Scholar, PGT Department of Electronics and Computer Science, R. T. M. Nagpur University, Nagpur.

P. K. Butey, Department of Computer Science, Kamla Nehru College, R.T.M. Nagpur University, Nagpur, India.

The fault prediction to estimate the software reliability is a challenging, interesting and important exercise, because software reliability is becoming more and more important in software industry. Various techniques are required to discover the faults in the development of software. However the reliability of software is measured in terms of failure and it is impossible to measure reliability before the software is completely developed. Software reliability is the most extensively studied quality among all the quality attributes [1]. The prediction of faults for estimating the software reliability is an essential requirement to know the global behaviour of the software and to ensure its proper working in future.

The software reliability generally depends upon the rate of failure in mean execution time or the number of faults occurred in execution time interval. The faults are likely to occur during the software design process i.e. from the requirements to realization. The number of faults increased as the size and complexity of the software increases. It is understood that as the more number of faults in the software, the rate of failure of software increases and the reliability of the software decreases. The faults that are introduced during the implementation are also considered as design faults. Generally it is not feasible to develop complex fault-free software, and even then, it is rarely feasible to guarantee that software is free of faults. Some formal methods can prove the correctness of software this means it matches to a specification document. However, today's formal verification techniques are not designed for the application to large software systems such as consumer operation systems or word processors. Furthermore, correctness does not ensure reliability because the specification document can itself be faulty. As it is not feasible to develop complex software systems free of faults and the absence of faults cannot be guaranteed, the reliability of software needs to be evaluated in order to fulfil high dependability requirements. Obviously, copies of (normal) software will fail together, if executed with the same parameters. This shows that the independence assumption does not hold. More precisely, the failure probabilities of software copies

are completely dependent. This makes many hardware fault tolerance principles ineffective for software. Instead of using redundant copies, software reliability can be improved by using design diversity. A common approach for this is the so called N-version programming, introduced by Chen and Avižienis [4, 5]. However, the research of Knight and Leveson [6] indicates, that design diversity is likely to be less effective for software than N-modular redundancy in hardware reliability engineering. Some studies have shown that for complex systems, the majority of failures are typically caused by software faults. Although software faults are design faults, their behaviour in dependable systems is similar to transient hardware faults. This is due to the stochastic of their activation conditions [7].

It is quite natural that to maintain the correct functionality of the software or fault free execution of software, the number of faults likely to be occurred in future execution time interval should be predicted and resolved. This prediction becomes essential for the component based software system, because the number of faults in complete software execution may or may not depend upon the number of faults in the components during the same execution time interval. Therefore for component based software model, the prediction of faults from execution of complete software must be observed with the prediction of number of faults of each component in the same execution time interval. This may suggest the number of faults likely to be occurred in future execution time interval may depend on the number of faults likely to be occurred in each component. In spite of different statistical approaches [8-10] for estimating the software reliability, techniques of Artificial Neural Network (ANN) are emerging as powerful tool for predicting the faults in future execution time interval for estimating the software reliability [11,12,13]. It has proven to be a universal approximates for any non-linear continuous function with an arbitrary accuracy [14, 15, 16, 17]. It has become an alternative method in software reliability modelling, evolution and prediction. Karunanithi, et al., [18] were the first to propose using neural network approach in software reliability prediction. Aljahdali, et. al., [19, 20], Adnan, et. al., [21], Park, et al., [22] and Liang, et.al., [15, 16] have also made contributions to software reliability predictions using neural networks, and have gained better results as compared to the traditional analytical models with respect to predictive performance. Karunanithi et al. [18] reports the use of neural networks for predicting software reliability, including experiments with both feed-forward and Jordan networks with a cascade correlation learning algorithm. Wittig and Finnie [23] describe their use of back propagation learning algorithms on a multilayer perception in order to predict development effort. An overall error rate (MMRE) obtained which compares favourably with other methods. Another study by Samson et al. [24] uses an Albus multilayer perception to predict software efforts on Boehm's COCOMO dataset. The work compares linear regression with a neural networks approach using the COCOMO dataset. But, both approaches seem to perform badly with MMRE of 52.7% and 42.1%, respectively. Srinivasan and Fisher [25] also report the use of a neural network with a back propagation learning algorithm. They found that the neural network outperformed other techniques and gave results up to 70%. However, it is not clear that how the dataset was divided for training and validation purposes. Khoshgoftaar et al. [26] presented a case study considering real time software to predict the testability of each module from source code static measures. They consider ANNs as promising techniques to build predictive models, because they are capable of modelling nonlinear relationships. The new trends in software engineering are for component based software design. Now it is important to estimate the reliability of component based software.

Component reliability can be derived using Software Reliability Models (SRMs), Software Reliability Growth Models (SRGM), or fault seeding models [27]. First, components should be tested and/or inspected separately before they are integrated with other components. The faults can be detected and tested in each component during component testing and/or inspection. Thus, reducing the number of faults must be managed during system testing. It is possible to estimate component reliability after predicting the number of faults for complete software and testing each component. The calculation for the prediction of number of faults for complete software based on the prediction of faults for the components does not produce an accurate prediction for the number of faults likely to be occurred in the software system. When components are integrated and interact with one another, new faults are expected to be triggered and the component reliabilities derived during component testing will diminish. However, the prediction for the number of faults of the software as estimated after the prediction of number of faults likely to be occurred in components provide an upper

limit for reliability for fault prediction. Since more faults are expected to be detected once components are integrated, the number of faults in the software system is expected to be no better than that derived from the component's faults prediction.

Most existing analytical methods to obtain reliability measures for software systems are based on the Markovian models [28, 29], and they rely on the assumption on exponential failed components time distribution. The Markovian models are subject to the problem of intractably large state space. Methods have been proposed to model reliability growths of components, which can be accounted for by the conventional analytical methods [30], but they are also facing the state space explosion problem. Discrete event simulation, on the other hand, offers an attractive alternative to analytical methods as it can capture a detailed system structure when performing software reliability analysis. Some simulation methods have been proposed, and a detailed description of simulation techniques for software reliability analysis and evaluation can be found in [31]. However, for component based software systems, it is difficult to analyse the influence to reliability caused by dependency among components. Based on the Shooman model [32, 33], a new structure based software reliability model is proposed. This model assumes that the software components have a high reliability, the component failures are independent, and the execution path failures are independent.

The reliability of a software product is usually defined to be "*the probability of execution without failure for some specified interval of natural units or time".* This is an operational measure that varies with how the product is used. Reliability of a component is measured in the context of how the component will be used. That context is described in an operational profile. Reliability of a piece of software may be computed or measured. If the software has not been built yet, its reliability can be computed from a structural model and the reliabilities of the individual parts that will be composed to form the software. There is error associated with this technique due to emergent behaviours, i.e. interaction effects, which arise when the components must work together. If the software is already assembled, the reliability can be measured directly. The measurement is taken by repeatedly executing the software over a range of inputs, as guided by the operational profile. The test results for the range of values are used to compute the probability of successful execution for a specific value. The error associated with this approach arises from the degree to which the actual operation deviates from the hypothesized operation assumed in the operational profile. Roshanak [34] discussed the uncertainty of the execution profile is modelled using stochastic processes with unknown parameters, the compositional approach to calculate overall reliability of the system as a function of the reliability of its constituent components and their (complex) interactions and sensitivity analysis to identify critical components and interactions will be provided. Lance Fiondella and Swapna S. Gokhale [35] considered the estimation of software reliability in the presence of architectural uncertainties and presented a methodology to estimate the confidence levels in the architectural parameters using limited testing or simulation data based on the theory of confidence intervals of the multinomial distribution. The sensitivity of the system reliability to uncertain architectural parameters was then quantified by varying the parameters within their confidence intervals. C. Smidts [36] presented architecturally based software reliability model and underlines its benefits. The models based on architecture derived from the requirements which captures both functional and non-functional requirements and on a generic classification of functions, attributes and failure modes. The model focuses on evaluation of failure mode probabilities and uses a Bayesian quantification frame work. Leslie Cheung and Leana Golubchik [37] discussed representative uncertainties which have identified at the level of a system's components, and illustrate how to represent them in reliability modelling framework.

In this paper, we are predicting the number of faults likely to be occurred in each component of the software as well as for the whole software for the future execution time interval using Component Based Neural Network Architecture (CBNNA). In this, we consider three layers of Feed Forward Neural Network Architecture (FFNNA) consisting of Input layer, Hidden layer and Output layer. In this architecture, the hidden layer consist with number of units correspond to the number of components in the software. Thus each unit of hidden layer itself is a separate Multilayer Feed Forward Neural Network Architecture (MFFNNA) consisting with three layers. Neural network architecture for each component is considered the execution time interval as an input pattern to the input layer of components and trained with *generalized delta learning rule* for the observed

number of faults in components. This process is considered for each component of the software. The output units of each component will provide the input to the output layer which is also a MFFNNA. In this neural network architecture, we have a input layer, two hidden layer and a output layer. The input layer considers the input pattern vector which is constructed from the simulated output of each component for the given execution time interval. This neural network considers the number of faults observed from the execution of complete software for the given execution time interval as target or object output pattern vector. This neural network is also trained with *generalized delta learning rule* and it is used to predict the expected number of faults in the future execution time interval. In this process, for the same execution time interval the CBNNA is trained for the observed faults of the complete software. Therefore on completion of training pattern from the training set the future execution time interval is presented as input pattern to the input layer for CBNNA and it is expected that this trained network will predict the number of faults likely to be occurred. Thus, our proposed CBNNA is expected to predict the number of faults for the complete software with the dependency of predicted faults in the each component of the component based software. The prediction for the expected number of faults in future execution time interval is also observed from each component and the predicted faults is presented as input pattern to the input layer of CBNNA to predict the faults from complete software for the same expected execution time interval. Thus, the faults prediction for the complete software depends upon the prediction of faults from each component. The simulated results indicate that the predicted faults for the whole software shows a weak dependency upon the predicted faults of components. Result also shows that the numbers of predicated faults from the complete software are less than total number of predicated faults of all components.

2. Literature Review

In recent years, many papers have presented various models for software reliability prediction [1]. In this section, the work related to neural network modelling for software reliability and prediction is presented. Numerous factors like software development process, organization, testing, software complexity, software faults and possibility of occurrence affect the software reliability. These factors represent non-linear pattern. Neural network methods normally approximate any non-linear continuous function. So, now-a-days, more attention is given to neural network based methods.

Karunanithi et al. [38, 14] first presented neural network based software reliability model to predict cumulative number of failures. They used execution time as the input of the neural network. They used different networks like feed forward neural networks, Jordan neural network and Elman neural network in their approach. They used two different training regimes like prediction and generalization in their study. They compared their results with statistical models and found better prediction. Karunanithi et al. [18] also used connectionist models for software reliability prediction. They applied the Falman's cascade Correlation algorithm to find out the architecture of the neural network. They considered the minimum number of training points as three and calculated the average error (AE) for both end point and next-step prediction. Their results concluded that the connectionist approach is better for end point prediction.

Sitte [13] presented a neural network based method for software reliability prediction. He compared the approach with recalibration for parametric models using some meaningful predictive measures with same datasets. They concluded that neural network approach is better predictors.

Cai et al. [39] proposed a neural network based method for software reliability prediction by using back propagation algorithm for training. They used multiple recent 50 failure times as input to predict the next-failure time as output and evaluated the performance of approach by varying the number of input nodes and hidden nodes. They concluded that the effectiveness of the approach generally depends upon the nature of the handled data sets.

Tian and Noore [15, 16] presented an evolutionary neural network based method for software reliability prediction by using multiple-delayed-input single output architecture. They used genetic algorithm to optimize the number of input nodes and hidden nodes of neural network.

Viswanath [40] proposed two models such as neural network based exponential encoding (NNEE) and neural network based logarithmic encoding (NNLE) for prediction of cumulative number of failures in

software. He encoded the input i.e. the execution time using the above two encoding scheme. He applied the approach on four datasets and compared the result of the approach with statistical models and found better results.

Hu et al. [41] proposed ANN model to improve the early reliability prediction for current projects/ releases by reusing the failure data from past projects/ releases. Su et al. [42] proposed a dynamic weighted combinational model (DWCM) based on neural network for software reliability prediction. They used different activation functions in the hidden layer depending upon the SRGM. They applied the approach on two data sets and compared the result with statistical models. The experimental result shows that DWCM approach provides better result than the traditional models.

Aljahdali et al. [43] investigated the performance of four different paradigms for software reliability prediction. They presented four paradigms like multi-layer perceptron neural network, radial-basis functions, Elman recurrent neural networks and a neuro-fuzzy model. They concluded that the adopted model has good predictive capability. All of the above mentioned models only consider single neural network for software reliability prediction. In [44], it was presented that the performance of a neural network system can be significantly improved by combining a number of neural networks. Jheng [45] presented neural network ensembles for software reliability prediction. He applied the approach on two software data sets and compared the results with single neural network model and statistical models. Experimental results show that neural network ensembles have better predictive capability.

In [46, 63] feed forward neural network for software reliability prediction is used. In this approach back propagation algorithm is applied to predict software reliability growth trend. The experimental result had shown that the proposed system has better prediction than some traditional software reliability growth models.

3. Component Based Software Prediction Models

Computer system plays a very important role in our daily lives as computer system failures can lead to a huge economic loss or even endanger human life. Reliability is one of the most important quality requirements of computer systems. A computer system comprises of hardware and software. The growing importance of software dictates that the software reliability is the major stumbling block in highly dependent computer system. Researchers have focused on procedural and object oriented software reliability. However, at present, there is a lack of similar research effort for Component Based Software (CBS). Furthermore, owing to certain specific features of CBS, existing reliability assessment frameworks for procedural or object oriented software cannot be applied as such to CBS. However, neither Black Box Statistical Software Testing (BBSST) nor any other existing testing models are capable of adequately support modern CBS development techniques. To support these techniques, testing models are needed to:

1. Explain the dependency of failure probability for software on its components.
2. Exploit reused software components of known reliability to estimate overall system reliability.

The above two points requires statistical models to describe the failure patterns for both individual software components and compositions of those components. The existing software reliability models for legacy systems are inappropriate for CBS. There is a need of such type of models which is based upon the system architecture. Many reliability models based upon the system architecture have been proposed. These models are known as architecture based reliability model, which may be used for the following reasons:

- To develop a method that analyses the application reliability built from the commercial off-the-shelf components (COTS) software components.
- To understand system reliability dependency on individual component reliabilities and their interconnection mechanism.

Swapna S. Gokhale [48] proposed some limitations for architecture based analysis technique. She classified the limitations into five categories namely modelling, analysis, parameter estimation, validation and optimization. Modelling limitations include concurrent execution, non-Markov transfer of control, non-exponential sojourn time, and interface failures etc. Analysis limitation includes reliability estimation, sensitivity and interface analysis, uncertainty quantification etc. Standard software engineering concept of a component is the basic

entity in the architecture based approach. A component is conceived as a logically independent entity of the system which performs a particular function. Component can be independently designed, implemented, and tested. User can define the component which depends on the factors such that probability of getting required data. Software architecture is the way of defining the software behaviour with respect to the manner in which different software components interact with each other. Beside this the failure behaviour is also associated with software architecture. Component failure behaviour can be expressed in terms of their reliabilities or failure rates.

Component Based Software Reliability Prediction

Component Based Software Engineering (CBSE) is a branch of software engineering which emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. Software engineers regard components as part of the starting platform for service-orientation. Components play this role, for example, in Web Services, and more recently, in Service-Oriented Architecture (SOA) – whereby a component is converted into a service and inherits further characteristics beyond that of an ordinary component. An individual component is a software package. To make a component based system the required components are put together, each component can communicate to other component via their interface characteristics.

Many of the researchers have proposed the various models for the estimation for the prediction of component based software reliability. The number of techniques for estimating the reliability of CBS system has been proposed. In order to estimate the reliability of CBS system, it is required to measure the reliability of each component of the system. The reliability of the system is based on the reliabilities of the individual components and the system architecture. Jean Dolbec and Terry Shepard proposed a model that estimates software system reliability from the reliability of its components and the usage ratio of each component. The Shooman's execution path model when transformed into a component based model, defines the reliability of a system as a function of the reliability of the components and it's usage ratios.

In [49] provide a survey of the research techniques and performed an analytical study for the estimation of system reliability based on component reliability and architectures. In their approach they considered the attribute of reliability component and credibility value, i.e. the risk parameter associated with reliability and computed the reliability as the product of reliability of successful reading data with the probability of correct execution of the actual function provided by the component and the probability for successful writing of the data. They could not estimate the values of these components and the credibility parameter and remain as the limitations of their work for Modelling of component dependencies.

In [50] proposes an approach to analysing the reliability of the system based on the reliability of the individual components and architecture of the system. They assumed the failures of software components are independent and the transfer of control among software components follows a Markov process. They predicated the reliability of the system is the product of components reliability. They utilized the Markov process to model the failure behaviour of the applications. They have proposed three methodologies for estimating the reliability of software system. The limitations of their study were that they have not explained anything about to assess reliability of individual component and have not studied effect of component failure on system failure. No specific reason was given for the second assumption which contradicts the study for the dynamic systems.

In [51] a reliability assessment method is proposed for incorporating the interaction among software components based on Analytic Hierarchy Process (AHP) to consider the effect of each software component on the reliability of entire system under distributed development environment and concluded that, the logarithmic Poisson execution time model fits better than the other SRGM's for the actual data set. They have also compared the inflection S-shaped software reliability growth model and the other models based on a Non-Homogeneous Poisson Process (NHPP), applied to reliability assessment of the entire system. They have not explained how AHP works in this area whether it requires some expert to use it, was the limitation here.

In [52] an integrated approach is proposed for modelling and analysing of component based systems with multiple levels of failures and faults recovery both at the software as well as the hardware level. Then they have used this approach to analyse overall

reliability. In their approach they encompass Markov Chain and queuing network modelling for estimating system reliability. They have extended the existing Discrete Time Markov Chain (DTMC) based reliability modelling techniques, modelling component based system with restarts and retries. Their assumptions of a failure at any software component will cause overall failure is not realistic and component retries is not true were turn on to the limitations.

In [53] a method is proposed to calculate the reliability of entire software system based on Stochastic Petri Nets (SPN) that evaluates component software reliability at the early stages of software development by decomposing component software into several subsystems and analysing the reliability of every subsystem and then by applying combinational analysis method. It can describe the process of dynamic changes of software, and it also considers the factors that affect software reliability by analysing the characteristic of software architecture. They depend on Markov process to calculate the reliability. In [54] a new approach is proposed to evaluate the reliability of CBS in open distributed environment by analysing the reliabilities of the components in different application domains, the reliabilities of the connections to these components and the architecture style of their composition. He has done a detailed reliability analysis for components, connections, architecture styles and software system. He depends totally on the previous models without any modification to calculate component, connection and system reliability.

In [55] a framework is proposed for assessing reliability of individual component, model analysis for assessing reliability of software system and for extension of reliability assessment model to complex distributed software system. It came out with the limitation that it is not easy to apply.

In [56] overruled the basic assumption that the component failure does not necessarily cause a system failure unless and until the failure of that component propagate the error. They have associated each component with error propagation probability, propagation path probability; depending on that they proposed probabilistic model of a component-based system. It is difficult to assess the error propagation probability.
In [57] a methodology is based on the execution scenario analysis of the COTS component based software application proposed to help the developer and integrators to regain some control over their COTS components by predicting the upper and lower bound on the reliability of their application systems. The maximum and minimum reliability values are obtained from various execution scenarios. This model requires lot of effort. The reliability of component can be predicted only after its existence with available failure data and not during its development.

In [58] a reliability assessment method based on the neural network in terms of estimating the effect of each component on the entire system is proposed by estimating the weight parameter for each component from input-output rules of neural network. However it is difficult to apply, and require more effort.

In [59] a framework for reliability prediction of components at architecture level is proposed. They overcome the lack of operational profile information by utilizing a variety of other available information sources. Also they proposed an approach to use hidden Markov models. Their framework provides meaningful reliability prediction in the context of early stages of software development; but for obtaining that, it requires more development time, so that their framework is time consuming and requires more effort.

In [60] a reliability model has developed and a reliability analysis technique for architecture-based reliability valuation and developed an approach to improve the accuracy of model evaluation by improving the accuracy of component reliability as well as transition probability. But they have not explained how to apply the proposed model to real software application.

In [61] reliability prediction model of component-based software architectures depends on reliabilities of components and operational profiles of use case packages is proposed. Component Based Software Reliability Model (CBSRM) based on the reliabilities of resources and component: while a resource can fail at any time independent of its usages, so that, the reliability of a resource is a function of time, CBSRM based on operational profiles: by using the Markov model as basic to give prediction of reliability, they have developed a way to build operational profile which accounts for the influence of the input data and user behaviour.

4. Artificial Neural Network

The SRM are widely used to assess the software reliability. Selecting an appropriate model based on the characteristics of the software projects is challenging because most of the SRM possesses certain restrictions or assumptions. Basically two approaches both require users to manually opt for candidates are adapted to locate the suitable model. The first one is to design a guideline, which could suggest fitting models for software projects. The other is to select the one with the highest confidence after various assessments. The decision-making processes would be a huge overhead while the software projects are huge and complicated.

In order to reduce such overhead, researchers proposed an alternative approach that can adapt the characteristics of failure processes from the actual data set by using neural networks. For example, Karunaithi et al. [38] applied neural network architecture to estimate the software reliability and used the execution time as input, cumulative the number of detected faults as desired output, and encoded the input and output into the binary bit string. The results showed that the neural network approach was good at identifying defect-prone modules software failures. Khoshgoftaar et al. [47] ever used the neural network as a tool for predicting the number of faults in programs. They introduced an approach for static reliability modelling and concluded that the neural networks produce models with better quality of fit and predictive quality. In addition, Cai et al. [39] examined the effectiveness of the neural network approach in handling dynamic software reliability data overall and present several new findings. They found that the neural network approach is more appropriate for handling datasets with smooth trends than for handling datasets with large fluctuations and the training results are much better than the prediction results in general.

It is well known that the neural networks are learning mechanisms that can approximate any non-linear continuous functions based on the given data. In general, neural networks consist of three components mainly neuron, network architecture and learning algorithm. Each neuron can receive signal, process the signals and finally produce an output signal. Figure 1 depicts a neuron, where f is the activation function that processes the input signals and produces an output of the neuron, x are the outputs of the neurons in the previous layer, and w are the weights connected to the neurons of the previous layer.

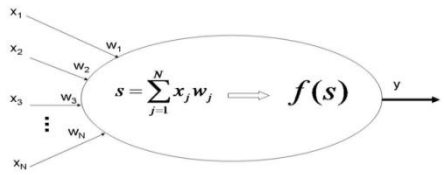

Figure 1: Artificial neuron.

The most common type of neural network architecture is called feed-forward network as shown in Figure 2. This architecture is composed of three distinct layers, i.e. an input layer, a hidden layer, and an output layer. In this figure the circles are represented as neurons and the connection of neurons across layers is called connecting weight. The back-propagation learning algorithm or the generalized delta learning rule describes the process to adjust the weights of neural network. During the learning processes, the weights of network are adjusted to reduce the errors of the network outputs as compared to the objective answers. In back-propagation algorithm, the weights of the network are iteratively trained with the errors propagated back from the output layer.

Subsequently, we describe the learning algorithm of the neural network in a mathematical form. The objective of the neural networks is to approximate an non-linear function that can receive the vector

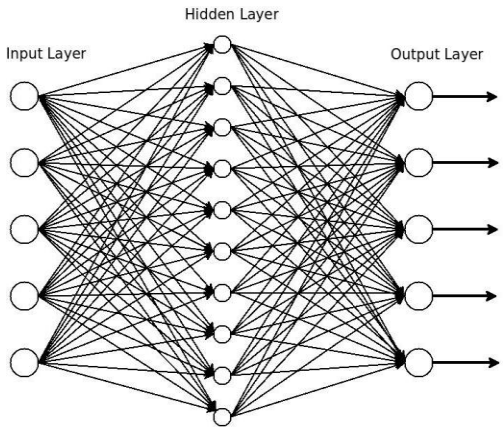

Figure 2: Neural Network Architecture for a Feed Forward Network.

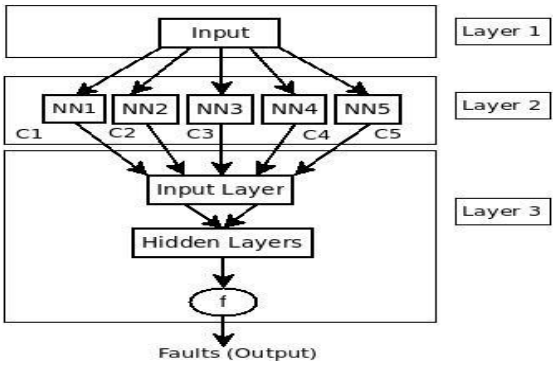

Figure 3: Design of proposed component based Neural Network Architecture (CBNNA).

$(x_1,x_2,...,x_n)$ in R^N and output the vector $(y_1, y_2,...,y_m)$ in R^M. Thus, the network can be denoted as:

$y = F(x)$, (1)

Here $x = (x_1, x_2,...,x_n)$ and $y = (y_1, y_2,...,y_m)$. The values of y_k are given as:

$$y_k = g(b_k + \sum_{j=1}^{H} w_{jk}^0 h_j), k = 1,.....,M, \quad (2)$$

Where w_{jk}^0 is the output "weight" from hidden layer node j to output layer node k, b_k is the bias of the node k, h_j is the output of the hidden layer, and g is an activation function in output layers. The values of the hidden layer is given by

$$h_j = f(b_j + \sum_{i=1}^{N} w_{ij}^1 x_i), j = 1,.....,H, \quad (3)$$

Where w_{ij} is the input "weight" from input layer node i to hidden layer node j, b_j is the bias of the node in the hidden layer, x_i is the value at input node i, and f is the activation function in hidden layer. The approximated function of the neural networks can be considered as some compound functions, which can be rewritten as nested functions, such as $f(g(x))$. Because of this feature, the neural network can be applied to software reliability modelling since software reliability modelling is likely to build a model to explain the software failure behaviour.

5. Implementation & Simulation Design

The reliability prediction in software engineering for any type of software whether simple software, modular software and CBS, mostly depends upon failure intensity. The failure intensity is itself depends upon so many factors. The prominent factor is on which failure intensity directly proportional to the number of faults likely to be occurred in mean execution time interval. For any software if the numbers of faults are observed in given execution time interval then the failure intensity of software can be easily determined. The major issue for estimating the generalized failure intensity from the predicted number of faults in future execution time interval is an important phenomenon in software reliability. There are several methods in the literature has been proposed as we have discussed in previous section. The new trends for predicting the failure rate or faults with the help of ANN are also applied in various manners. The major contribution to use ANN for predicting the number of faults is mainly for the simple software structure. Here our concern is to apply ANN paradigms for predicting the number of faults likely to be occurred in future execution time interval for CBS. It is quite natural to understand that for any CBS the number of faults in the software depends upon the number of faults occurring in each component. It is also obvious that in a particular execution time interval if components are presenting more number of faults then in the same time interval the whole software will also exhibit more number of faults. Thus, we can say that the number of faults in entire software is directly depending upon the number of faults occurs in the components of the software.

Now it is very important to select a particular model and technique for software reliability assessment. But sometimes software projects cannot fit the assumption of a unique model. Therefore to overcome from this problem, Lyu and Allen [62] have proposed a solution by combining the results of different software reliability models. This approach inspired us to use the NN based approach to combine the fault prediction assessment for each component of the software for predicting the number of faults assessment for complete software. Now we present our proposed approach to accomplish the task of predicting the numbers of faults occur in any CBS.

In our proposed method we consider a Component Based Feed Forward Neural Network Architecture (CBFFNNA) which consists with three layers as shown in figure 3. The first layer is an Input layer which is presented with input pattern in the form of execution time interval as $(\Delta t_1, \Delta t_2,.....,\Delta t_n)$ and corresponding output patterns are in the terms of number of faults observed from the execution of complete software in the respected execution time

interval Δt. i.e. *(F_1, F_2,.....,F_n)* . Thus the training set for the complete software consist with input-output pattern pairs. Now we present the input pattern vector to the second layer of our model. The second layer consists with the number of components in the software. In this layer each component is a feed-forward neural network. Thus if we have *n* number of components in the software, then we have *n* neural network architectures as *NC_1, NC_2, NC_3,.....,NC_i*. Each neural network architecture corresponding to component say *NC_i* for the *i^{th}* component consist with three layers i.e. Input layer, hidden layer and output layer. Thus we can represent the neural network architecture for the *i^{th}* component as *$NC(x_i,h_i,y_i)$*. Therefore, the output for the *i^{th}* component of neural network architecture can be shown as:

$$NC_i(x_i)_i = \Delta t_i \quad (4)$$

$$NC_i(h_i)_h = f\left[\sum_{h=1}^{H} w_{hi}\Delta t_i\right] \quad (5)$$

$$NC_i(y_i)_j = f\left[\sum_{j=1}^{J} w_{jh} NC_i(h_i)_h\right] \quad (6)$$

Let the number of faults observed from the *i^{th}* component are *(Cf_1, Cf_2 ,...Cf_n)*. It is considered as the target output for the *i^{th}* component i.e. *NC_i* to train this neural network with back propagation learning rule. Here we consider back propagated error in the terms of least-mean-square as:

$$E_{NC_i} = \frac{1}{2}\sum_{j=1}^{J}(Cf_j - NC_i(y_i)_j)^2 \quad (7)$$

The generalized data learning rule is used to update weight vector for the *i^{th}* component neural network architecture. The weight update can be presented as

$$\Delta W_{jh}\alpha - \frac{\partial ENC_i}{\partial w_{ih}} \quad (8)$$ And,

$$\Delta W_{hi}\alpha - \frac{\partial ENC_i}{\partial w_{hi}} \quad (9)$$

Hence every component provides the simulated output after the training for each execution time interval of training set. Let for execution time interval Δt_1 ...Δt_n each component say *i^{th}* provides the simulated output as:

$$(SNC_i^1, SNC_i^2, ..., SNC_i^n) \quad (10)$$

Now we consider input pattern vector to train the neural network for complete software i.e. CBFFNNA. The input pattern vector for the neural network of complete software is of order *$j\times1$*; so that now we select the maximum value of component output among the components simulated output for execution time interval Δt_i as:

$$P_i = \max(SNC_i^1, SNC_i^2, ..., SNC_i^n)_{\Delta t_1} \quad (11)$$

Therefore, in this way we construct the input pattern vector to the input layer of neural network architecture of complete software as:

$$P = [P_{1(1toN)}^T P_{2(1toN)}^T .....P_{i(1toN)}^T] \quad (12)$$

Hence, the input pattern vector *P* is applied to the input layer of the neural network for the complete software. This neural network is constructed with two hidden layers and one output layer. The target output pattern vector to accomplish the training is the number of faults observed from the execution of software for the given execution time interval i.e. (*F_1, F_2, ...,F_n*). The output from each hidden layer and output layer for this neural network architecture can be represented as:

$$CSW_{h1} = f[\sum_{h_1}^{H_1} w_{h_1 i} P_i] \quad (13)$$

$$CSW_{h2} = f[\sum_{h_2=1}^{H_2} w_{h_1 h_2} h_1] \quad (14)$$

$$CSW_o = f[\sum_{o=1}^{n} w_{oh_2} CSW_{h_2}] \quad (15)$$

Again we consider the generalized the delta learning rule to train this neural network. The weights are modified according to gradient descent approach to minimize the mean square error as:

$$E^F = \frac{1}{2}\left[\sum_{o=1}^{n}(F_o - csw_o)\right]^2 \quad (16)$$

$$\Delta W_{oh_2}\alpha - \frac{\partial E^k}{\partial W_{oh_2}} \quad (17)$$

$$\Delta W_{h_1 h_2}\alpha - \frac{\partial E^k}{\partial W_{h_1 h_2}} \quad (18)$$

$$\Delta W_{h_1 i}\alpha - \frac{\partial E^k}{\partial W_{h_2 i}} \quad (19)$$

In our experiment, we consider a component based software system which consists with 5 components say *C_1, C_2, C_3, C_4, C_5*. Now, we observed the execution of software for the execution time from *t_0 to t_{99}*. We also observed the number of faults during the execution from time *t_0 to t_{49}* as represented in table 1.

Table 1: Training set in the form of Input-Output Pattern pairs for complete software

Execution time interval	Execution time	Input to first layer (complete software)	Observed faults in Dec.& Bin.	
$t_0 - t_4$	Δt_1	000100	5	00101
$t_5 - t_9$	Δt_2	000101	12	01100
$t_{10} - t_{14}$	Δt_3	001110	7	00111
$t_{15} - t_{19}$	Δt_4	001111	17	10001
$t_{20} - t_{24}$	Δt_5	011000	11	01011
$t_{25} - t_{29}$	Δt_6	011001	9	01001
$t_{30} - t_{34}$	Δt_7	100010	6	00110
$t_{35} - t_{39}$	Δt_8	100011	18	10010
$t_{40} - t_{44}$	Δt_9	101100	14	01110
$t_{45} - t_{49}$	Δt_{10}	101101	10	01001

In this table 1, we obtained the execution time interval by considering the Exclusive OR (XOR) operation for the time step of 5. The execution time interval is in gray code. The observed faults are obtained in decimal numbers and converted them in binary code. Thus, our training set represents the input-output pattern pairs in which the inputs is of 6 bit gray code representing the execution time interval and the output pattern is of 5 bit representing the observed fault. In a same manner, we consider the training set for the neural network architectures of components as in table 2.

Table 2: Training set in the form of Input-Output Pattern pairs for Components

Execution time interval	Observed faults in components									
	FC_1		FC_2		FC_3		FC_4		FC_5	
	Dec	Bin	Dec	Bin	Dec	Bin	Dec	Bin	Dec	Bin
Δt_1	2	010	0	000	1	001	0	000	2	010
Δt_2	4	100	2	010	1	001	7	111	2	010
Δt_3	0	000	3	011	2	010	2	010	0	000
Δt_4	5	101	3	011	2	010	6	110	1	001
Δt_5	2	010	6	110	0	000	2	010	1	001
Δt_6	3	011	4	100	2	010	0	000	0	000
Δt_7	1	001	2	010	1	001	0	000	2	010
Δt_8	6	110	4	100	3	011	2	010	3	011
Δt_9	0	000	5	101	3	011	6	110	0	000
Δt_{10}	1	001	2	010	3	011	2	010	1	001

In this table 2, the input pattern is considered as execution time interval and the observed faults for each component in binary code. Now the input-pattern from Δt_1 to Δt_{10} is presented to the input layer of our model as shown in figure 3. The neural network architectures for the components are started for training with their respected observed faults i.e. FC_1 to FC_5. The rest of the execution time interval from t_{11} to t_{20} are considered as test pattern set for predicting the expected number of faults likely to be occurred from each component as well as from complete software.

6. Results and Discussion

In our experiment, the component based software with 5 components and Δt1 ….Δt10 execution time interval is considered to train the neural networks to each component. The number of observed faults from each component i.e. FC1 to FC5 are used as target output pattern for components. Thus for each execution time interval, each component has its own observed faults. Hence all the 5 neural network architecture corresponding to all components are trained with trainlm training function of neural network available in matlab. The performance for each component is shown in figures 4, 5, 6,7, and 8.

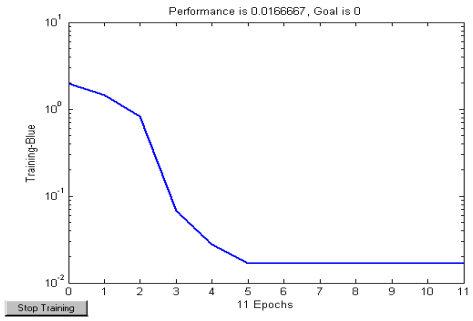

Figure 4: Training performance for First Component (FC1).

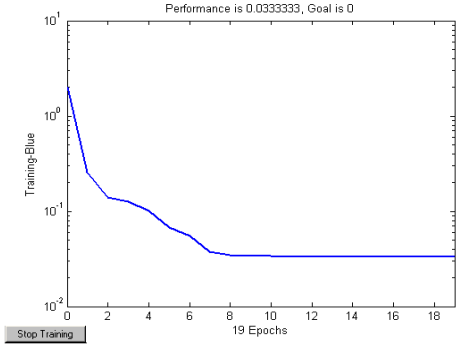

Figure 5: Training performance for Second Component (FC2) .

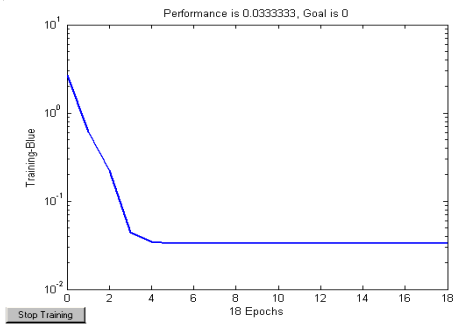

Figure 6: Training performance for Third Component (FC3).

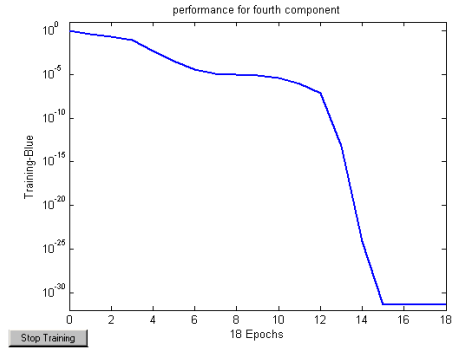

Figure 7: Training performance for Fourth Component (FC4).

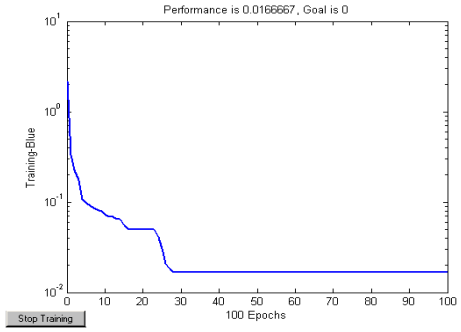

Figure 8: Training performance for Fifth Component (FC5).

The training performances are found satisfactory for first component (C_1), fourth component (C_4) and fifth component (C_5), whereas for the other components the performances are found near to convergence. After the training, the execution time interval Δt_1 to Δt_{10} is presented as input to verify the performance of these trained neural networks. The simulated output is presented in table 3 as:

Table 3: Simulated Output from Components on given execution time interval

input	FC_1	FC_2	FC_3	FC_4	FC_5
Δt_1	2	0	1	0	2
Δt_2	4	2	1	7	2
Δt_3	0	3	2	2	0
Δt_4	5	3	2	6	1
Δt_5	2	6	0	2	1
Δt_6	1	2	1	0	0
Δt_7	1	2	1	0	2
Δt_8	6	4	3	2	3
Δt_9	0	5	3	6	0
Δt_{10}	1	2	3	2	1

The difference between simulated faults and actual observed faults for the execution time interval from Δt_0 to Δt_{10} or each component can be seen in figures 9, 10,11,12,13 respectively.

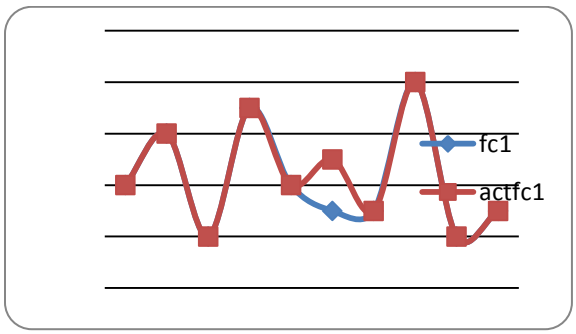

Figure 9: Performance evaluation for Component FC1.

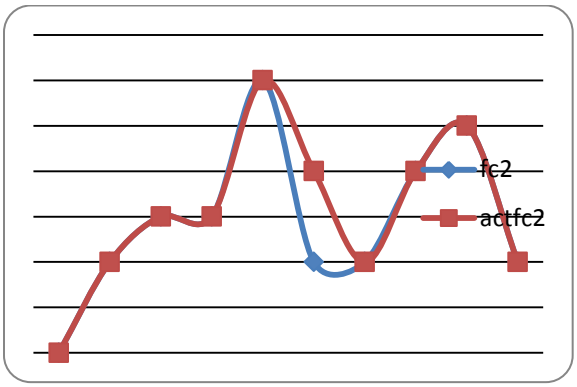

Figure 10: Performance evaluation for Component FC2.

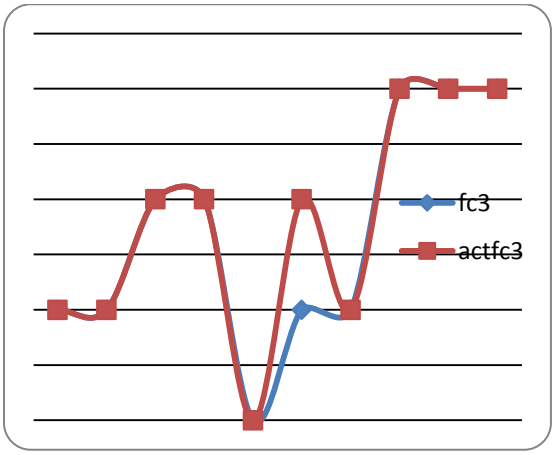

Figure 11: **Performance evaluation for Component FC3.**

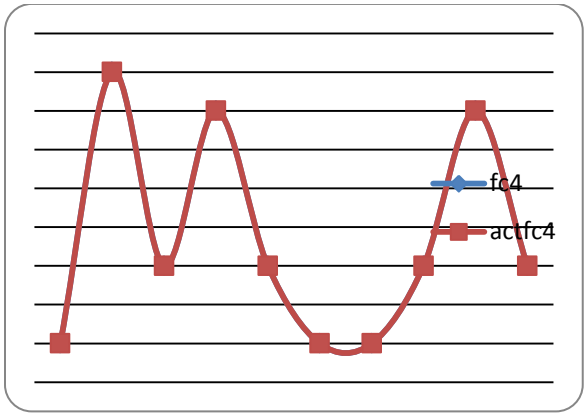

Figure 12: **Performance evaluation for Component FC4.**

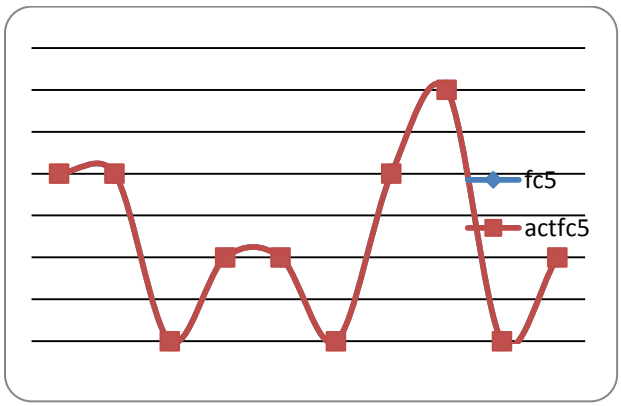

Figure 13: **Performance evaluation for Component FC5.**

Now these simulated faults are used as input pattern for the third layer of our model as shown in figure 3. The neural network architecture for our third layer consists with input layer of 5 neurons, 2 hidden layers with 20 and 10 neurons respectively. The output layer consists with 5 neurons, because our target output pattern in terms of observed faults for whole software is of 5 bits. This neural network is also trained with training algorithm *trainlm* of feed-forward neural network in *matlab*. The performance of training is depicted in figure 14.

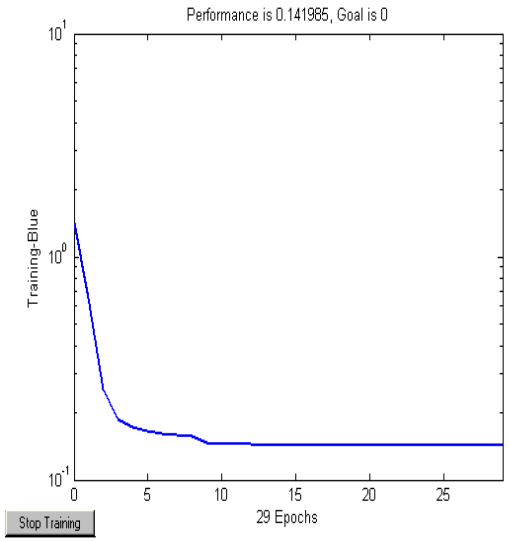

Figure 14: Training performance for complete software.

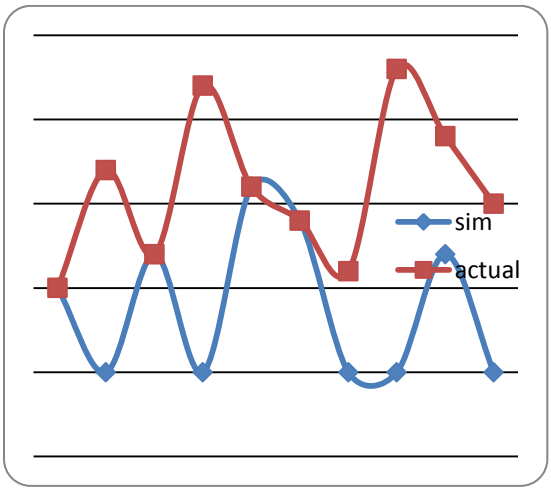

Figure 15: **Performance evaluation for complete software.**

It is found that the neural network could not be completely converged for the given training set but the performance is near to satisfactory. After this we present the execution time interval from time Δt_1 to Δt_{10} as input to verify the performance of this trained neural network. The comparison between the simulated output and actual output are shown in table 4 and in figure 15 as:

Table 4: Simulated Output from Complete Software for given execution time interval

Input	Simulated Output	Actual Output
Δt_1	5	5
Δt_2	0	12
Δt_3	7	7
Δt_4	0	17
Δt_5	11	11
Δt_6	9	9
Δt_7	0	6
Δt_8	0	18
Δt_9	7	14
Δt_{10}	0	10

The combine and comparative performance evaluation of neural networks for components and the neural network for complete software is represented in table 5 and table 6. The table 5 is presenting the variance in between simulated output and actual output for the components and whole software whereas the table 6 is representing the standard deviation in between simulated output and actual output for the components and whole software. The same performance can also be observed from figure 16 for variance and figure 17 for standard deviation.

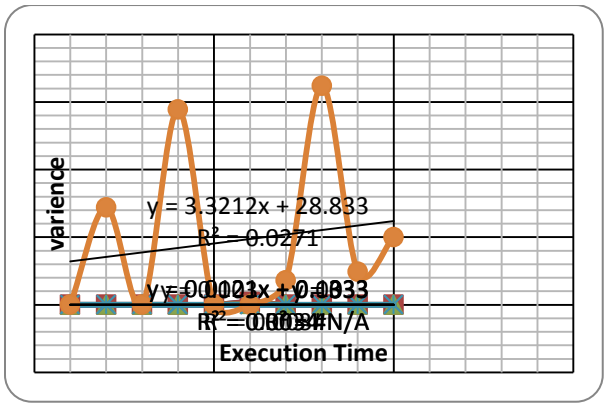

Figure 16: Variance in between Performance evaluation of Components and complete software for training set.

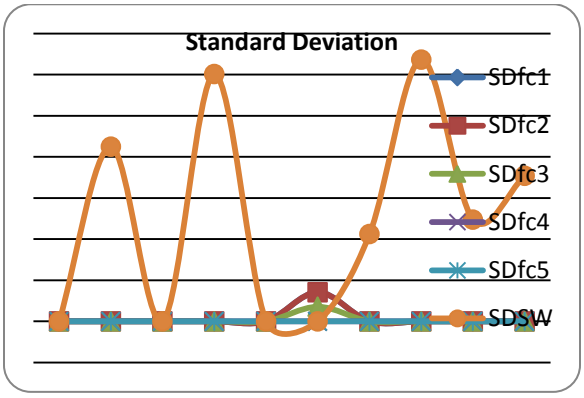

Figure 17: Standard Deviation in between Performance evaluation of Components and complete software for training set.

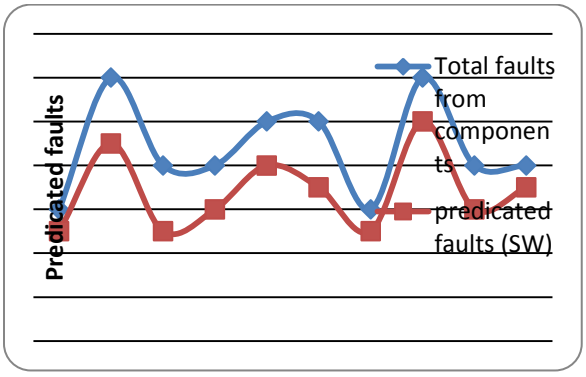

Figure 18: Total number of Predicated faults in components and Complete Software.

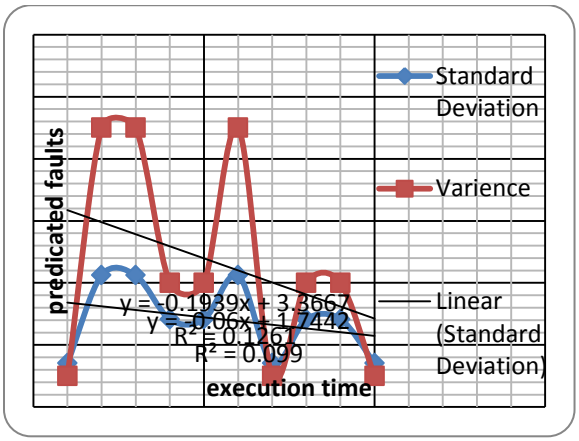

Figure 19: Standard Deviation and Variance in between predicated faults from components and complete software.

Table 5: performance evaluation of Components and complete software for training set.

Simulated output							Actual output						Variance					
input	fc1	fc2	fc3	fc4	fc5	simcsw	actfc1	actfc2	actfc3	actfc4	actfc5	actswf	varfc1	varfc2	varfc3	varfc4	varfc5	varSw
Δt_1	2	0	1	0	2	5	2	0	1	0	2	5	0	0	0	0	0	0
Δt_2	4	2	1	7	2	0	4	2	1	7	2	12	0	0	0	0	0	72
Δt_3	0	3	2	2	0	7	0	3	2	2	0	7	0	0	0	0	0	0
Δt_4	5	3	2	6	1	0	5	3	2	6	1	17	0	0	0	0	0	144.5
Δt_5	2	6	0	2	1	11	2	6	0	2	1	11	0	0	0	0	0	0
Δt_6	1	2	1	0	0	9	3	4	2	0	0	9	2	2	0.5	0	0	0
Δt_7	1	2	1	0	2	0	1	2	1	0	2	6	0	0	0	0	0	18
Δt_8	6	4	3	2	3	0	6	4	3	2	3	18	0	0	0	0	0	162
Δt_9	0	5	3	6	0	7	0	5	3	6	0	14	0	0	0	0	0	24.5
Δt_{10}	1	2	3	2	1	0	1	2	3	2	1	10	0	0	0	0	0	50

Table 6: Standard Deviation for performance evaluation of components and complete software for training. set

Simulated output							Actual output						Standard Deviation					
time	fc1	fc2	fc3	fc4	fc5	simcsw	actfc1	actfc2	actfc3	actfc4	actfc5	actswf	SDfc1	SDfc2	SDfc3	SDfc4	SDfc5	SDSW
Δt_1	2	0	1	0	2	5	2	0	1	0	2	5	0	0	0	0	0	0
Δt_2	4	2	1	7	2	0	4	2	1	7	2	12	0	0	0	0	0	8.485281
Δt_3	0	3	2	2	0	7	0	3	2	2	0	7	0	0	0	0	0	0
Δt_4	5	3	2	6	1	0	5	3	2	6	1	17	0	0	0	0	0	12.02082
Δt_5	2	6	0	2	1	11	2	6	0	2	1	11	0	0	0	0	0	0
Δt_6	1	2	1	0	0	9	3	4	2	0	0	9	1.4142	1.414214	0.707107	0	0	0
Δt_7	1	2	1	0	2	0	1	2	1	0	2	6	0	0	0	0	0	4.242641
Δt_8	6	4	3	2	3	0	6	4	3	2	3	18	0	0	0	0	0	12.72792
Δt_9	0	5	3	6	0	7	0	5	3	6	0	14	0	0	0	0	0	4.949747
Δt_{10}	1	2	3	2	1	0	1	2	3	2	1	10	0	0	0	0	0	7.071068

Table 7: Predication of faults for Components and complete software for expected execution time interval.

Time interval	Predicted faults from components						Predicted faults from s/w	Standard Deviation	Variance
	fc1	fc2	fc3	fc4	fc5	total			
Δt_{11}	2	2	2	0	0	6	5	0.707106781	0.5
Δt_{12}	2	0	2	4	4	12	9	2.121320344	4.5
Δt_{13}	0	2	2	0	4	8	5	2.121320344	4.5
Δt_{14}	2	0	2	4	0	8	6	1.414213562	2
Δt_{15}	2	2	2	0	4	10	8	1.414213562	2
Δt_{16}	0	0	2	4	4	10	7	2.121320344	4.5
Δt_{17}	2	2	2	0	0	6	5	0.707106781	0.5
Δt_{18}	2	0	2	4	4	12	10	1.414213562	2
Δt_{19}	0	2	2	0	4	8	6	1.414213562	2
Δt_{20}	2	0	2	4	0	8	7	0.707106781	0.5

Now, we consider our test pattern set from execution time interval Δt_{11} to Δt_{20} and predict the number of faults from each component. The predicted faults from each component for the execution time interval from t_{11} to t_{20} are now considered as test input pattern for the whole software to predict the number of faults from the software. The prediction of faults from each component and from the whole software is presented in table 7. This Table considers the predicted faults from each component, total faults around components and predicted faults from software. The number of predicted faults from each component for every execution time interval are added and presented in

the column of this table as total faults from components. It is compared with predicted faults from complete software.

It is observed that the total faults from components are approximately seen as predicted faults from whole software with small deviation. The standard deviation and variance in between total predicted faults from components and predicted faults from software are presented in last two columns of the table. The performance in between total predicted faults from components and predicted faults from whole software can be seen in figure 18. The figure 19 is representing standard deviation and variance in between total predicted faults from components and predicted faults from whole software. Thus the observed simulated results are showing that the total predicted faults from the complete software are approximately less than the total predicted faults from the components.

7. Conclusion

In our experiment for component based software with 5 components and Δt1 ….Δt10 execution time interval is considered to train the neural networks for each component. The number of observed faults from each component i.e. FC1, to FC5 are used as target output pattern for components. Thus for each execution time interval, each component has its own observed faults. Hence all the 5 neural network architecture corresponding to all components are trained with trainlm training function of neural network available in matlab. The performance for each component is depicting from figure 4,5,6,7, and 8. The estimation of software reliability for any type of software, whether the modular software, object oriented software, depends upon man factors. The factor which influences the estimation is determined from the number of faults occurs in the software during its course of execution. Thus, to maintain the correct functionality of software or fault free execution of software, the number of expected faults in expected execution time interval should be predicted correctly and efficiently. This prediction of faults becomes more essential for the component based system because the number of faults in software execution directly depends upon the number of faults in the components of software. Therefore to understand the necessity for predicting the faults during the execution of software, we considered in this paper a model for predicting the expected number of faults for future execution time interval for each component of software and also for the complete software. Hence in our approach, we consider a CBNNA which consists with three layers. The first layer i.e. input layer considers the input pattern vector in terms of the execution time interval Δt. The second layer i.e. hidden layer consider the number of units same as the number of components in the software. Thus each unit represents separate neural network architecture of three layers. Hence each neural network architecture corresponds to each component considers the input pattern as execution time interval and the output pattern as the observed fault in that component. Each neural network trained and after the training produces the simulated output as simulated faults. These simulated faults are used as input pattern for the third layer of our proposed CBNNA. The third layer is another a feed forward neural network of 4 layers i.e. input layer, 2 hidden layers and output layer. The input layer considers the simulated faults from the previous hidden layer as input pattern. The observed fault during the course of software execution is considered as target or object output pattern for these inputs. The last layer neural network architecture is also trained with generalized delta learning rule and after the training it produces the simulated output in terms of the faults. The execution time interval from Δt11 to Δt20 is used as future execution time interval and presented as input to the CBNNA. Each component of the software predicts the expected faults and as well as for the whole software. We conducted the experiment on software which consist with large 5 components and obtained the simulated results. The following observations are being made during the simulation design and implementation.

1. The simulated behavior of each component for the given execution time interval is found approximately same with their actual observed faults. The simulated faults from the component 4 and 5 are found exactly same with the actual observed faults without any deviation.
2. The simulated faults from the complete software are not found approximately to the actual observed faults in the given execution time interval, but the deviation is not higher. It is also observed that the simulated faults from whole software are found less than the actual number of faults observed in the software. The reason of this behavior is being that the numbers of faults in components are not increasing the number

of faults for the complete software execution, because particular fault behavior of the component could not participate in the execution of complete software.

3. The expected number of faults during the future execution time interval from Δt_{11} to Δt_{20} is estimated from each component as well for the complete software. It is being observed that after summation of these predicted faults of the components is found more than the number of predicted faults for the complete software. Again the same reason which has been stated earlier is verified that the number of faults in the component may be more than the number of faults found during the course of execution of software. There is more investigation and analysis is required to justify our observations on complex software that consists with large number of components.

Acknowledgment

We wish to acknowledge the guidance and support from Dr. P. K. Butey, Dr. Manu Pratap Singh and Head, PGT Department of Electronics and Computer Science, R. T. M. Nagpur University, Nagpur.

References

[1] Michael R. Lyu , *Handbook of Software Reliability Engineering*, McGraw-Hill publishing, 1995, ISBN 0-07-039400-8, http://portal.research.belllabs.com/orgs/ssr/book/ reliability/introduction.html.

[2] "IEEE Std 610.12-1990", IEEE *Standard Glossary of Software Engineering Terminology*, 1990.

[3] J. Tian, "Software Quality Engineering", John Wiley and Sons Inc. 2005.

[4] A. AVIZIENIS, "On the Implementation of N-Version Programming: A Fault-Tolerance Approach during Execution," COMPSAC 77, Chicago, IL, 1977, pp. 149-155.

[5] L. CHEN, and A. AVIZIENIS, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," Proceedings of FTCS-8, Toulouse, France, 1978, pp. 3–9.

[6] I. Eusgeld, F. Fraikin, M. Rohr, F. Salfner and U. Wappler, "Software Reliability", Dependability Metrics, LNCS-4909, Springer-verlag Berlin Heidelberg, pp. 104-125, 2008.

[7] N. D. Singpurwalla and S.P. Wilson. Statistical Methods in Software Engineering. Springer Series in Statistics. Springer-Verlag, New York, 1999.

[8] Jonh D. Musa, A. Lanino and K. Okumoto, Software Reliability Measurement, Prediction and Application, McGraw-Hill, 1987.

[9] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision3, Naval Software Weapons Center, Revised September 1993.

[10] Aljahdali, S., Sheta, A., and Rine, D., "Prediction of Software Reliability: A Comparison between regression and neural network non-parametric Models" Proceeding of the IEEE/ACS Conference, pp. 470-471, 2000.

[11] Hu Q., Xie M., and Ng S., Software Reliability Predictions using Artificial Neural Networks, Computational Intelligence in Reliability Engineering (SCI) 40, 197-222, 2007.

[12] K. Y. Cai, L. Cai, W. D. Wang, Z. Y. Yu, and D. Zhang, "On the Neural Network Approach in Software Reliability Modeling", The Journal of Systems and Software, 2001, pp. 47-62.

[13] R. Sitte, "Comparison of software–reliability-growth predictions: neural networks vs parametric recalibration", IEEE Transactions on Reliability, vol. 48, no. 3, pp. 285- 291, 1999.

[14] N. Karunanithi, D. Whitley, Y.K. Malaiya, "Using neural networks in reliability prediction", IEEE Software, vol. 9, no. 4, pp.53–59, 1992.

[15] L. Tian, A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction, Reliability Engineering and System Safety", vol. 87, no. 1, pp. 45–51, 2005.

[16] L. Tian, A. Noore, "On-line prediction of software reliability using an evolutionary connectionist model", The Journal of Systems and Software, vol.77, no. 2, pp. 173–180, 2005.

[17] 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", vol. 14, no.1, pp. 79–88, 2003.

[18] N. Karunanithi, D. Whitley, Y.K. Malaiya, "Prediction of software reliability using connectionist models", IEEE Trans Software Engg., vol. 18, no. 7, pp. 563-573, 1992.

[19] S.H. Aljahdali, D.Rineand A.Sheta, "Prediction of software reliability: A comparison between regression and neural network non-parametric models", Computer Systems and Applications, ACS/IEEE International Conference on, 0:0470, 2001.

[20] K. A. Buraggaand, S. Aljahdali, "Evolutionary neural network prediction for software reliability modeling", In The 16th International Conference on Software Engineering and Data Engineering (SEDE-2007), 2007.

[21] W. A. Adnan and M. H. Yaacob, "An integrated neural-fuzzy system of software reliability

prediction", In Proc. Conf. First Int Software Testing, Reliability and Quality Assurance, pp.154–158, 1994.

[22] J.-H. Park J.-Y. Park and S.-U. Lee, "Neural network modeling for software reliability prediction from failure time data" Journal of Electrical Engineering and information Science, vol. 4, no.4, pp. 533–538, 1999.

[23] G. Witting, and G. Finnie, "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort", J. Information Systems, vol. 1, no. 2, pp. 87-94, 1994.

[24] B. Samson, D. Ellison, and P. Dugard, "Software Cost Estimation Using Albus Perceptron (CMAC)," Information and Software Technology, 1997, vol.39, pp. 55-60. 1997.

[25] K. Srinivazan, and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, February, vol.21, no.2, pp.126-137, 1995.

[26] T. M. Khoshgoftaar, E.B. Allen, and Z. Xu, "Predicting testability of program modules using a neural network," Proc. 3Rd IEEE Symposium on Application-Specific Systems and Software Engg. Technology, pp. 57-62, 2000.

[27] J. Dolbec, "A Structure Based Software Reliability Model", M. E. dissertation, RMC, May 1995.

[28] R. C. Cheung. "A User-Oriented Software Reliability Model". IEEE Trans. on Software Engineering, SE-6(2):118–125, March 1980.

[29] J. C. Laprie and K. Kanoun. Handbook of Software Reliability Engineering, M. R. Lyu, Editor, chapter Software Reliability and System Reliability, pages 27–69. McGraw-Hill, New York, NY, 1996.

[30] S. Gokhale and K. S. Trivedi. "Structure-Based Software Reliability Prediction". In Proc. of Fifth Intl. Conference on Advanced Computing, Chennai, India, December 1997.

[31] Musa, J. D., and Okumoto, K., "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," Proceedings seventh International Conference on Software Engineering, Orlando, Florida, pp. 230-238, 1984.

[32] M. L. Shooman, "Software Engineering: Design, Reliability and Management", McGraw Hill, ISBN 0-07-057021-3, 1983.

[33] M.L. Shooman, "A Micro Software Reliability Model for Prediction and Test Apportionment", Proceedings 1991 International Symposium on Software Engineering (Austin, Texas), pp. 52-59, May 1991.

[34] Roshanak Roshandel, "Calculating Architectural Reliability via Modeling and Analysis.", A Dissertation Presented to the Faculty of the Graduate School University of Southern California, Los Angeles, CA 90089-0781 U.S.A., December 2006.

[35] Derek Doran, Matthew Tran, Lance Fiondella, and Swapna S. Gokhale, "Architecture-based Reliability Analysis with Uncertain Parameters." *SEKE*. 2011.

[36] Smidts, C., and D. Sova. "An architectural model for software reliability quantification: sources of data." *Reliability Engineering & System Safety* 64.2 (1999): 279-290.

[37] Cheung, Leslie, Leana Golubchik, Nenad Medvidovic, and Gaurav Sukhatme. "Identifying and addressing uncertainty in architecture-level software reliability modeling." In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1-6. IEEE, 2007.

[38] N. Karunanithi, Y.K. Malaiya, D. Whitley, "Prediction of software reliability using neural networks", Proceedings of the Second IEEE International Symposium on Software Reliability Engineering, pp. 124–130, 1991.

[39] K.Y. Cai , L. Cai , W.D. Wang , Z.Y. Yu , D. Zhang, "On the neural network approach in software reliability modeling", The Journal of Systems and Software, vol. 58, no. 1, pp. 47-62, 2001.

[40] Bisi, Manjubala, and Goyal Neeraj Kumar. "Software Reliability Prediction using Neural Network with Encoded Input." *International Journal of Computer Applications* 47 (2012).

[41] Q.P. Hu, Y.S. Dai, M. Xie, S.H. Ng, "Early software reliability prediction with extended ANN model", Proceedings of the 30th Annual International Computer Software and Applications Conference, pp. 234-239, 2006.

[42] Y.S. Su, C.Y. Huang, "Neural-Networks based approaches for software reliability estimation using dynamic weighted combinational models", The Journal of Systems and Software, vol. 80, no. 4, pp. 606-615, 2007.

[43] S.H. Aljahdali, K.A. Buragga, "Employing four ANNs paradigm for Software Reliability Prediction: an Analytical study", ICGST International Journal on Artificial Intelligence and Machine Learning, vol. 8, no. 2, pp. 1-8, 2008.

[44] P.M. Granotto, P.F. Verdes, H.A. Caccatto, "Neural network ensembles: Evaluation of aggregation algorithms, Artificial Intelligence", vol. 163, no.2, pp. 139-162, 2005.

[45] J. Zheng, "Predicting Software reliability with neural network ensembles, Expert systems with applications", vol. 36, no. 2, pp. 2116-2122, 2009.

[46] Y. Singh, P. Kumar, "Application of feed-forward networks for software reliability prediction", ACM SIGSOFT Software Engineering Notes, vol. 35, no. 5, pp. 1-6, 2010.

[47] T. M. Khoshgoftaar, R. M. Szabo, and P. J. Guasti, "Exploring the Behavior of Neural-network Software Quality Models," Software Eng. J., Vol. 10, no. 3, pp. 89–96, May 1995.
[48] [48] S. Gokhale et al., "Architecture Based Software Reliability Analysis: Overview and Limitations", IEEE Transactions on Dependable and Secure Computing. pp 32-40, 2007.
[49] Aleksandar Dimov and Sasikumar Punnekkat, "On the Estimation of Software Reliability of Component Based Dependable Distributed systems", In R.Reussner et al, eds. QoSA-SOQUA. A Bibliographic Sourcebook, pp. 171-187. Berlin: Springer Verlage, 2005.
[50] Jung-Hua Lo, Chin-Yu Hung, Ing-Yi Chen, Sy-Yen Kuoand Michael R.Lyu, "Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure," the journal of systems and software., Vol. 76, pp. 3-13, 2005.
[51] Yoshinobu Tamura and Shigeru Yamada, "Comparison of Software reliability Assessment methods for Open Source Software and Reliability Assessment Tool", Journal of Computer Science, Vol. 2, No. 6, pp. 489-495, 2006.
[52] Vibhu Saujanya Sharma and Kishor S.Trivedi, "Reliability and Performance of Component Based Software Systems with Restarts, Retries, Reboots and Repairs." 17th International Symposium on Software Reliability Engineering (ISSRE'06), 2006.
[53] Lianzhang Zhu and Yanchen Li, "Reliability Analysis of Component Software Based on Stochastic Petri Nets," in Proc. 6th IEEE/ACIS International Conference on Computer and Information Science, 2007.
[54] Haiyang Hu, "Reliability Analysis for Component- based Software System in Open Distributed Environments," IJCSNS International Journal of Computer Science and Network Security., Vol. 7, No. 5, pp. 193- 202, May 2007.
[55] R. Amuthakkannan, S.M. Kannan, K. Vijayalakshmi and V. Jayabalan, "Managing change and reliability of distributed software system," International Journal of Information Systems and Change Management, Vol. 2, No. 1, pp. 30-49, 2007.
[56] Vittorio Cortellessa and Vincenzo Grassi, "A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems", CBSE 2007, July, 2007.
[57] Tirthankar Gayen and R.B Misra, "Reliability Bounds Prediction of COTS Component Based Software Application." IJCSNS International Journal of Computer Science and Network Security, Vol. 8, No. 12, pp. 219-228, December 2008.
[58] Yoshinobu Tamura and Shigeru Yamada, "Component-Oriented Reliability Analysis and Optimal Version-upgrade Problems for Open Source Software", Journal of Software, Vol. 3, No. 6, pp. 1-8, June 2008.
[59] Leslie Cheung, Roshanak Roshandel, Nenad Medvidovic and Leana Golubchik, "Early Prediction of Software Component Reliability," in Proc. ICSE 2008, May 2008.
[60] Fan Zhang, Xingshe Zhou, Junwen Chen and Yunwei Dong, "A Novel Model for Component-based Software Reliability Analysis," in Proc. 11th IEEE High Assurance Systems Engineering Symposium., pp. 303- 309, 2008.
[61] Pham Thanh Trung and Huynh Quyet Thang, "International Journal of Information Technology." Vol. 5, No. 1, pp. 18-25, 2009.
[62] M. R. Lyu and A. Nikora, "Using Software Reliability Models more effectively," IEEE Software, pp. 43-52, 1992.
[63] Nidhi Gupta and Manu Pratap Singh, "Estimation of Software Reliability with Execution time model using Pattern mapping Technique of Artificial Neural Network", Computer and Operation Research, ELESVIER, vol. 32, pp. 187-199, 2005.

Deepak Shudhalwar, M. Sc., M. Phil. in Computer Science, is a Research Scolar, persuing Ph.D. in Computer Science, in the Department of Elecronics and Computer Science, R.T.M. Nagpur University, Nagpur, India. Presently, he is working as a Assistant Professor in CSE, Department of Engineering & Technology, PSSCIVE, NCERT, Bhopal, India. His research area includes software reiability, artificial neural network and soft computing.

Dr. P. K. Butey, M. Sc., Ph. D. in Computer Science is a Research Guide in Computer Science, Department of Elecronics and Computer Science, R.T.M. Nagpur University, Nagpur, India. Presently, he is working as a Head, Department of Computer Science, Kamla Nehru College, Nagpur, India. His research area includes software reiability, artificial neural network, fuzzy logic and soft computing.