# Prediction of Software Faults Using Fuzzy Nonlinear Regression Modeling

Zhiwei Xu
Taghi M. Khoshgoftaar*
Florida Atlantic University
Boca Raton, Florida USA

Edward B. Allen
Mississippi State University
Mississippi USA

## Abstract

*Software quality models can predict the risk of faults in modules early enough for cost-effective prevention of problems. This paper introduces the Fuzzy Nonlinear Regression (FNR) modeling technique as a method for predicting fault ranges in software modules.* FNR *modeling differs from a classical linear regression in that the output of an* FNR *model is a fuzzy number. Predicting the exact number of faults in each program module is often not necessary.* FNR *model can predict the interval that the number of faults of each module falls into with a certain probability.*

*A case study of a full-scale industrial software system was used to illustrate the usefulness of* FNR *modeling. This case study included four historical releases. The first release's data were used to build the* FNR *model, the remaining three releases' data were used to evaluate the model. We found that* FNR *modeling gives useful results.*

Keywords: *Software reliability, fuzzy nonlinear regression model, software metrics, neural networks, multiple linear regression*

## 1. Introduction

Software is often a key component of the high technology systems that are so common in modern society. Highly reliable software is crucial both to software producers and users, as well as society in general, because failures of software can cause major disruptions to business and can even threaten emergency service.

The field of software metrics is predicated on the idea that measurement of product attributes during development will give insight into the quality of operational software [1]. Prior research has shown that software product and process metrics collected prior to the test phase can be the basis for fault predictions. We want to enhance the quality of modules recommended by a model early enough to prevent problems from faults later in the life cycle because it is much more cost-effective to correct software faults early in the development process than later when they cause failure.

Predicting the exact number of faults in each program module is often not necessary. Previous research has focused on classification models to identify fault-prone and not fault-prone modules [9, 10]. Such models are based on the definition of fault-prone before modeling, usually via a threshold on the number of faults expected. However, available information is often uncertain, imprecise and incomplete. Before reliability problems become evident, it is difficult to choose an appropriate definition of fault-prone at the time of modeling. In such cases, predicting the range of the numbers of faults is more appropriate. A predicted interval consists of the possible minimum and maximum numbers of faults in the modules. A software manager might use this information to allocate testing efforts more effectively.

Our preliminary work showed that interval prediction using Fuzzy Linear Regression Modeling gave us new insights in quality prediction [18]. This paper extends the investigation to nonlinear models using Fuzzy Nonlinear Regression, FNR,[1] modeling. *Fuzzy* indicates that the output of the model is a fuzzy set; *nonlinear* means that the neural networks are used; and *regression* implies that quantities are predicated from a fit data set, similar to multiple linear regression.

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/.

[1] All acronyms represent either singular or plural according to the context.



0-7695-0927-4/00 $10.00 © 2000 IEEE

In this paper, two neural networks are used to build an FNR model, one neural network is trained to approach the upper limits of the dependent variable, and the other one estimates lower limits. These two neural networks are trained independently. Because the FNR introduced by Ishibuchi and Tanaka [3] are very sensitive to outliers and the training processes are not smooth, we propose a new type of FNR which is different from other FNR in that instead of focusing on a cost function, the mean of the dependent variable is used to build the FNR. The expected spreads of dependent variable values are determined prior to the training process, and the spreads are adjusted according to the training results.

A case study of a full-scale industrial software system was used to illustrate the usefulness of FNR for predicting faults. This case study included four historical releases. The first release's data set was used to build the FNR, the remaining three releases' data sets were used to evaluate the model. We found that FNR gives useful results.

The reminder of this paper summarizes our modeling methodology, fuzzy nonlinear regression, evaluating model accuracy, results of a full-scale industrial case study, and conclusions.

## 2. Modeling

Our goal is to develop a model that will predict the likely range of the number of faults. The number of faults can only be measured after software has become operational. In contrast, software metrics can be measured during development. A suitable software quality model makes predictions when it is not too late to take compensatory actions. The following is a summary of our methodology for building and validating the quality models in the context of a case study [11].

1. Preprocess measurements to improve the model. We use principal component analysis to reduce the dimensionality of the data and to produce orthogonal independent variables.

2. Select a model validation strategy, such as resubstitution, data splitting, cross-validation, or multiple projects [6]. Model validation evaluates the accuracy of a model.

3. Prepare *fit* and *test* data sets according to the model validation strategy. The *fit* data sets are used to build a model. The *test* data set is used to evaluate its accuracy [16].

4. Build an FNR model based on the *fit* data set. The number of faults is the dependent variable and software metrics are independent variables.

5. Evaluate the FNR model using *test* data sets by comparing predictions to the actual number of faults.

The following sections discuss fuzzy nonlinear regression modeling, and evaluating accuracy. Principal components analysis is summarized in the appendix.

### 2.1. Fuzzy Nonlinear Regression

Fuzzy sets [20], introduced by Zadeh in 1965 as a mathematical way to represent vagueness in linguistics, can be considered a generalization of classical set theory. A classical (*crisp*) set, $A$, has elements that are either members or not. A fuzzy set is a set that allows partial membership. The notion of membership in fuzzy sets thus becomes a matter of degree, which is a number between zero and one. A membership of degree zero represents complete nonmembership, while a membership of degree one represents complete membership. Thus, the membership function of a fuzzy set maps each element $x$ of the universe of discourse $\mathcal{U}$ into the interval [0 1].

A fuzzy set $A$ in the universe of discourse $\mathcal{U}$ can be defined as a set of ordered pairs.

$$A = \{(x, \mu_A(x)) \mid x \in \mathcal{U}\} \tag{1}$$

where $\mu_A(x)$ is the membership function of $x$ in $A$, which indicates the degree that $x$ belongs to $A$.

Whereas there exist numerous shapes of membership functions, we use one of the most common in practice, triangles. A triangular membership function of fuzzy set $A$ is specified by three parameters $(L, C, R)$ for the left, center, and right points. If the triangle is symmetric, we can use a pair to represent a fuzzy set as following.

$$A = (a^C, a^W) \tag{2}$$

Where $a^C$ and $a^W$ represent the center and spread of a triangular membership function [15].

With any fuzzy set $A$ we can associate a collection of sets known as $h$-levels [13] of $A$. An $h$-level is a classical set consisting of elements of $A$ which belong to the fuzzy set at least to a degree $h$. The *h-level* of a fuzzy set $A$ denoted as $[A]_h$ is defined for $0 < h \leq 1$ as follows.

$$[A]_h = \{x | \mu_A(x) \geq h, x \in \mathcal{U}\} \tag{3}$$

If the membership function of fuzzy set $A$ is symmetric triangular, the following is usually used to calculate the

interval $[A]_h$.

$$[A]_h = [a^C - (1-h)a^W, a^C + (1-h)a^W] \quad (4)$$

A fuzzy number [17] $A$ is a fuzzy set over the domain of real numbers which is normal [14] and convex [17]. In fuzzy set theory, a fuzzy set is *normal* when the height (the maximum value of its membership function) of the fuzzy set is one. The intuitive meaning of convexity is that the membership function of a convex fuzzy set does not go "up-and down" more than once. The outputs of FNR in our study are triangular fuzzy numbers.

Ishibuchi and Tanaka [3] proposed a concept for interval regression analysis based on back-propagation neural networks. One network identifies the upper side of data intervals and the other the lower side of data intervals. The purpose of this method is to obtain a fuzzy nonlinear regression model. They used an approximation approach to implement the training [3].

However, their approach focused on selecting a stepwise cost function for two neural networks (lower side and upper side). This algorithm is very sensitive to outliers, because the outliers cause only polarities of training data to be considered. Therefore we proposed a new algorithm based on the mean of the dependent variable $y_i$, which is less sensitive to outliers. Instead of seeking different cost functions like many researchers, we rely on the mean of the dependent variable to calculate a reasonable interval before training, and then the upper and lower limit of the interval will be the goals of the two independent neural networks. These two neural networks can use any training algorithm that is available, and the training can be smooth and very accurate.

Suppose $\{\mathbf{x}_i \mid i = 1, 2, ..., n\}$ and $\{y_i \mid i = 1, 2, ..., n\}$ are the independent and dependent variables, respectively, in the training data of the neural networks. The training data can be represented by $n$ input-output pairs $(\mathbf{x}_i, y_i)$, where $i = 1, 2, ..., n$. The independent variable vector $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{im})$ is of size $m$ while the dependent variable $y_i$ is of size one. Let $y_+(\mathbf{x}_i)$ and $y_-(\mathbf{x}_i)$ be the outputs from two neural networks corresponding to the independent variable vector $\mathbf{x}_i$, where each network has $m$ input nodes and a single output node. Our goal is to make the outputs $y_+(\mathbf{x}_i)$ and $y_-(\mathbf{x}_i)$ approximately satisfy the following condition:

$$y_-(\mathbf{x}_i) \le y_i \le y_+(\mathbf{x}_i) \quad (5)$$

The key point here is how to determine a reasonable upper and lower limit of the interval prior to training process. In other word, we need to obtain the spread $(y_i^W)$ of $y_i$ in advance such that $y_i \in (y_i - y_i^W, y_i + y_i^W)$, and then the lower side neural network will be trained to approach $y_i - y_i^W$ while the upper side neural network will approach $y_i + y_i^W$ with the same independent variable vector $\mathbf{x}_i$. The width $y_i^W$ should meet the following requirements:

- Positive number
- As small as possible, but subject to neural network accuracy.
- Reflecting the distribution of $y_i$.

In order to calculate the $y_i^W$, we need to observe the mean and standard deviation of $y_i$ as follows.

$$\bar{y} = \frac{\sum_{i=1}^n y_i}{n} \quad (6)$$

$$s_y = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}} \quad (7)$$

The reason we treat the mean in a special manner is we assume that the dependent variable values $y_i$ cluster around the mean $\bar{y}$, where oscillation of the neural network is likely to occur. This is not always true. For some data sets, dependent variable value may not cluster around one certain point so that this approach may not be suitable. This is an open issue that needs to be explored in the future. In our experience, dependent variable values often cluster around a certain point. So, we must give the $y_i$ around $\bar{y}$ a relatively large spread so that the oscillation of the neural network can be attenuated. According to the above discussion, we define the spread $y_i^W$ as following,

$$y_i^W = \frac{\phi}{\bar{y}} y_i + f(y_i) \quad (8)$$

Where $\phi$ is a constant and is interpreted as the spread of $y_i$ at the mean point. Empirical study found that $0.5\,\bar{y} \le \phi \le 0.7\,\bar{y}$ works well [19]. Small $\phi$ can't guarantee Equation (5) will be satisfied, and on the other hand, an extremely large $\phi$ results in a large total spread which may turn the interval into nonsense. Take the grade point average (GPA) [2] of a student which ranges from 0 to 4.0 in USA as an example. It is not useful to say somebody's (GPA) is between 0 to 4.0. The compensation function $f(y_i)$ is used to dampen the oscillation of the neural network output around $\bar{y}$. It is defined as following.

$$f(y_i) = \begin{cases} \frac{\psi y_i}{1+e^{-\theta_1(y_i-\omega_1)}} & y_i \le \omega_1 \\ \psi y_i & \omega_1 \le y_i \le \omega_2 \\ \frac{\psi y_i}{1+e^{-\theta_2(y_i-\omega_2)}} & y_i \ge \omega_2 \end{cases} \quad (9)$$

[2] In a course, students can get $A$=4.0, $B$=3.0, $C$ = 2.0, $D$ = 1.0, or $F$ = 0

where $\psi$ is a scale number, in our study we used $\psi = 1$. The parameter $\psi$ indicates how much attenuation we need for the oscillation area. The parameters $\omega_1$ and $\omega_2$ are the lower and upper bound of oscillation area which are obtained experimentally. The parameters $\theta_1$ and $\theta_2$ indicate how steep the two sides of the function will go down, $\theta_1 = \theta_2$ implies the function is symmetric. We define $\theta_1$ and $\theta_2$ as follows.

$$\theta_1 = \frac{\bar{y} - \omega_1}{s_y} \tag{10}$$

$$\theta_2 = \frac{\omega_2 - \bar{y}}{s_y} \tag{11}$$

Figure 1 and Figure 2 show two neural network training results comparing before and after adding the compensation function $f(y_i)$ in Equation (9). In these two figures, the horizontal axis is the module number and vertical axis is the number of faults in a module. In order to identify the oscillation region, we should sort modules by $y_i$ in ascending order before drawing the graph. The middle curve is the desired output $y_i$. The upper curve is the output of upper side of neural network $y_+(\mathbf{x}_i)$, while the lower curve is the output of lower side of neural output $y_-(\mathbf{x}_i)$. Before adding the compensation function (Figure 1), $y_i$, $y_+(\mathbf{x}_i)$, and $y_-(\mathbf{x}_i)$ interweave together in the oscillation area, and thus it is difficult to guarantee that Equation (5) is satisfied. After adding the compensation function (Figure 2), $y_+(\mathbf{x}_i)$ and $y_-(\mathbf{x}_i)$ are separated in the oscillation area so that Equation (5) is satisfied.

Because the two neural networks are trained independently, it may occur that the training results in $y_-(\mathbf{x}_i) > y_+(\mathbf{x}_i)$. If so, the interval model is meaningless for module $i$. To eliminate this abnormality, we introduce two functions $z_-(\mathbf{x}_i)$ and $z_+(\mathbf{x}_i)$ [14].

$$z_-(\mathbf{x}_i) = \begin{cases} y_-(\mathbf{x}_i) & y_-(\mathbf{x}_i) \le y_+(\mathbf{x}_i) \\ \frac{1}{2}[y_-(\mathbf{x}_i) + y_+(\mathbf{x}_i)] & y_-(\mathbf{x}_i) > y_+(\mathbf{x}_i) \\ 0 & y_-(\mathbf{x}_i) < 0 \end{cases} \tag{12}$$

$$z_+(\mathbf{x}_i) = \begin{cases} y_+(\mathbf{x}_i) & y_-(\mathbf{x}_i) \le y_+(\mathbf{x}_i) \\ \frac{1}{2}[y_-(\mathbf{x}_i) + y_+(\mathbf{x}_i)] & y_-(\mathbf{x}_i) > y_+(\mathbf{x}_i) \end{cases} \tag{13}$$

Therefore, $z_-(\mathbf{x}_i) \le z_+(\mathbf{x}_i)$ holds for all $\mathbf{x}_i$, the nonlinear interval $[z_-(\mathbf{x}_i), z_+(\mathbf{x}_i)]$ is always meaningful. We can derive a fuzzy nonlinear model from the interval $[z_-(\mathbf{x}_i), z_+(\mathbf{x}_i)]$. What we predict is the range of the number of faults in each module, this number cannot be negative, so we need to truncate the negative $z_-(\mathbf{x}_i)$ to zero before we build FNR. Let $\widehat{Y}(\mathbf{x}_i)$ be a fuzzy number that predicts dependent variable $y_i$. According to Equation (2) we can use symmetric triangular membership functions to implement the FNR.

$$\widehat{Y}(\mathbf{x}_i) = (f^C(\mathbf{x}_i), f^W(\mathbf{x}_i)) \tag{14}$$

From Equation (4), the $h$-level set of $\widehat{Y}(\mathbf{x}_i)$ is calculated by the following interval.

$$\begin{aligned} [\hat{Y}(\mathbf{x}_i)]_h &= [f^C(\mathbf{x}_i) - (1-h)f^W(\mathbf{x}_i), \\ &\quad f^C(\mathbf{x}_i) + (1-h)f^W(\mathbf{x}_i)] \end{aligned} \tag{15}$$

where $f^C(\mathbf{x}_i)$ and $f^W(\mathbf{x}_i)$ are the center and the spread of $\widehat{Y}(\mathbf{x}_i)$, respectively. We need to derive the fuzzy model $\widehat{Y}(\mathbf{x}_i)$ such that the $h$-level set of $\widehat{Y}(\mathbf{x}_i)$ satisfies the following condition for a given $h \in (0, 1]$.

$$[\widehat{Y}(\mathbf{x}_i)]_h = [z_-(\mathbf{x}_i), z_+(\mathbf{x}_i)], \text{ for all } \mathbf{x}_i \tag{16}$$

Let $[\widehat{Y}(\mathbf{x}_i)]_h^+$ be the upper limit of $[\widehat{Y}(\mathbf{x}_i)]_h$, let $[\widehat{Y}(\mathbf{x}_i)]_h^-$ be its lower limit, and let $[\widehat{Y}(\mathbf{x}_i)]_h^C = f^C(\mathbf{x}_i)$ be its center. From Equations (14) and (15), we can derive $f^C(\mathbf{x}_i)$ and $f^W(\mathbf{x}_i)$ for a given $h \in (0, 1]$.

$$f^C(\mathbf{x}_i) = \frac{1}{2}(z_-(\mathbf{x}_i) + z_+(\mathbf{x}_i)) \tag{17}$$

$$f^W(\mathbf{x}_i) = \frac{1}{2h}(z_+(\mathbf{x}_i) - z_-(\mathbf{x}_i)) \tag{18}$$

Thus, by Equation (14), we have a fuzzy number $\widehat{Y}(\mathbf{x}_i)$ that predicts the number of faults $y_i$ for module $i$ based on its software metrics $\mathbf{x}_i$.

In summary, we can take the following steps to build a fuzzy nonlinear regression model.

1. Calculate the mean ($\bar{y}$) and standard deviation ($s_y$) of the dependent variable $y_i$.
2. Select an appropriate $\phi$, for example, 50% to 70% of the mean $\bar{y}$ to scale the dependent variable's width.
3. Determine a spread $y_i^W$ using Equation (8) without a compensation function $f(y_i)$.
4. Calculate $y_i + y_i^W$ and $y_i - y_i^W$ as upper limit and lower limit of the interval.
5. Train two neural networks to approach the upper limit and lower limit.
6. If there is an oscillation area:
   (a) Identify oscillation area $y_i \in [\omega_1, \omega_2]$.
   (b) Calculate compensation function $f(y_i)$ using Equations (9)-(11). (The default value for $\psi$ is one.)

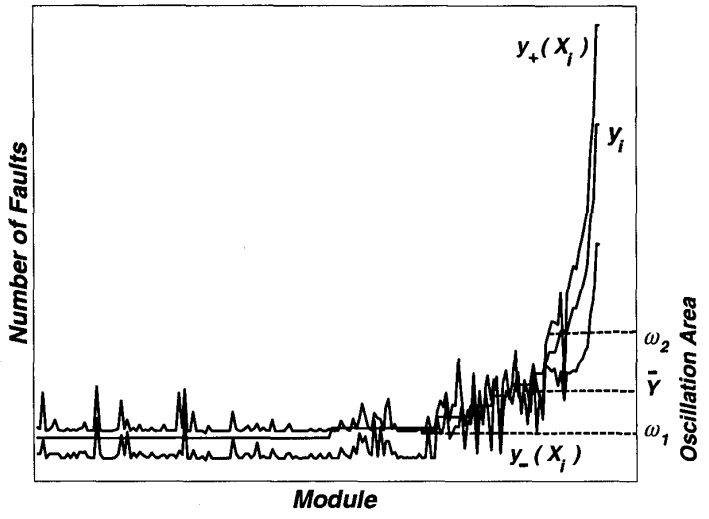


**Figure 1. Training result before adding compensation function $f(y_i)$**

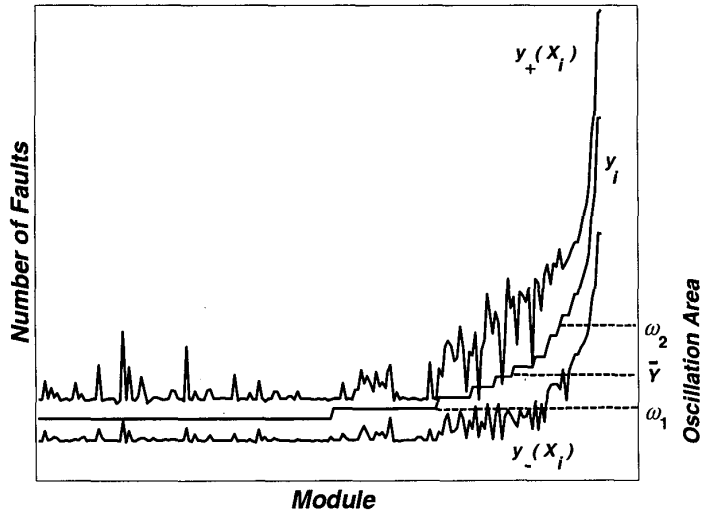


**Figure 2. Training result after adding compensation function $f(y_i)$**

(c) Add the compensation function $f(y_i)$ in Equation (8) to calculate $y_i^W$ and determine new upper limit and lower limit.

(d) Train two neural networks again with new upper and lower limit.

7. Calculate $z_-(\mathbf{x}_i)$, $z_+(\mathbf{x}_i)$ using Equation (12) and Equation (13)

8. Choose various $h$ values of interest and calculate $f^C(\mathbf{x}_i)$ and $f^W(\mathbf{x}_i)$ for each using Equation (17) and Equation (18).

9. Calculate $h$-level set of $\widehat{Y}(\mathbf{x}_i)$ using Equation (15).

### 2.2. Evaluating Accuracy

Several statistics quantify a model's accuracy on *fit* and *test* data [12]. Average absolute error ($AAE$) is a commonly used one which we apply here.

We have three different types of $AAE$ for our model, $AAE^+$, $AAE^-$, and $AAE^C$, they are associated with upper limit, lower limit, center of the $[\widehat{Y}(\mathbf{x}_i)]_h$ respectively.

$$AAE^* = \frac{1}{n}\sum_{i=1}^{n} |y_i - [\widehat{Y}(\mathbf{x}_i)]_h^*| \qquad (19)$$

where $*$ stands for $+$, $-$ or $C$.

$AAE^+$ explains how close the upper limit of $[\widehat{Y}(\mathbf{x}_i)]_h$ is to $y_i$. $AAE^-$ explains how close the lower limit of $[\widehat{Y}(\mathbf{x}_i)]_h$ is to $y_i$. $AAE^C$ explains how close the center of $[\widehat{Y}(\mathbf{x}_i)]_h$ is to $y_i$.

Let $AA^W$ be the average absolute width of $[\widehat{Y}(x)]_h$.

$$AA^W = \frac{2}{n}\sum_{i=1}^{n} |f^W(\mathbf{x}_i)| \qquad (20)$$

$AA^W$ explains how wide the interval is for a given $h$-level.

## 3. Empirical Study

### 3.1. System Description

We studied a large legacy telecommunication system written in a high-level language (Protel) similar to Pascal [7]. The metrics used in this case study were collected by the Enhanced Measurement for Early Risk Assessment of Latent Defects (EMERALD) system which is a decision-support system including software measurement facilities and software quality models [2].

Table 1 lists the names and descriptions of twenty-four product metrics used in our study [8]. Table 2 lists

**Table 1. Software Product Metrics**

| Symbol | Description |
|---|---|
| **Call Graph Metrics** | |
| *CALUNQ* | Number of distinct procedure calls to others. |
| *CAL2* | Number of second and following calls to others. *CAL2* = *CAL*−*CALUNQ* where *CAL* is the total number of calls. |
| **Control Flow Graph Metrics** | |
| *CNDNOT* | Number of arcs that are not conditional arcs. |
| *IFTH* | Number of non-loop conditional arcs (i.e., if-then constructs). |
| *LOP* | Number of loop constructs. |
| *CNDSPNSM* | Total span of branches of conditional arcs. The unit of measure is arcs. |
| *CNDSPNMX* | Maximum span of branches of conditional arcs. |
| *CTRNSTMX* | Maximum control structure nesting. |
| *KNT* | Number of knots. A "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles. |
| *NDSINT* | Number of internal nodes (i.e., not an entry, exit, or pending node). |
| *NDSENT* | Number of entry nodes. |
| *NDSEXT* | Number of exit nodes. |
| *NDSPND* | Number of pending nodes (i.e., dead code segments). |
| *LGPATH* | Base 2 logarithm of the number of independent paths. |
| **Statement Metrics** | |
| *FILINCUQ* | Number of distinct include files. |
| *LOC* | Number of lines of code. |
| *STMCTL* | Number of control statements. |
| *STMDEC* | Number of declarative statements. |
| *STMEXE* | Number of executable statements. |
| *VARGLBUS* | Number of global variables used. |
| *VARSPNSM* | Total span of variables. |
| *VARSPNMX* | Maximum span of variables. |
| *VARUSDUQ* | Number of distinct variables used. |
| *VARUSD2* | Number of second and following uses of variables. *VARUSD2* = *VARUSD* − *VARUSDUQ* where *VARUSD* is the total number of variable uses. |

**Table 2. Software Execution Metrics**

| Symbol | Description |
|---|---|
| *USAGE* | Deployment percentage of the module. |
| *RESCPU* | Execution time (microseconds) of an average transaction on a system serving consumers. |
| *BUSCPU* | Execution time (microseconds) of an average transaction on a system serving businesses. |
| *TANCPU* | Execution time (microseconds) of an average transaction on a tandem system. |

four execution metrics [8]. Measurements were aggregated to the module level. A module was a set of source code files. The software metrics on four historical releases were collected.

Following our modeling methodology, we preprocessed our product metrics using principal components analysis. Our stopping rule selected six components that accounted for over 89% of the overall variance. This reduced the data from 24 to 6 dimensions, simplifying the modeling. Table 3 presents the factor pattern of six principal components extracted from twenty-four product metrics of Releases 1. The correlation between a principal component and a raw metric is presented in each cell. The highest value in each row of the Table 3 is bold. Data from each release was transformed accordingly.

### 3.2. Results

We used the first release as the *fit* data set to build models, and the remaining three releases as *test* data sets. The dependent variable of the model was the number of faults $y_i$ after unit test through operations.

The six domain metrics along with the four execution metrics were the independent variables which were used in our models. The *fit* data set contained 3649 modules.

The key point for the fuzzy nonlinear regression models is to select the spread $y_i^W$. The mean and standard deviation for the number of faults in *fit* data were $\bar{y} = 1.03$ and $s_y = 2.16$. We followed the steps described in Section 2.1 to build the FNR. We initially chose $\phi = 0.68 = 0.66\bar{y}$ and $\psi = 1$ (the default value). Experiments showed that the oscillation occurred in region $y_i \in [1, 10]$, that is $\omega_1 = 1$ and $\omega_2 = 10$. Therefore, a compensation function was introduced to dampen the oscillation. After $\bar{y} = 1.03$, $s_y = 2.16$, $\omega_1 = 1$, $\omega_2 = 10$ along with initial $\phi$, and $\psi$ were brought to Equations (8)-(11), we determined the expected spread

**Table 4. Summary of Neural-Networks for the case study**

| | |
|---|---|
| **Architecture** | |
| Class | Perceptron |
| Connections | Feedforward |
| Layers | 5 |
| Input nodes | 10 |
| Hidden nodes | 40;40;40 |
| Output nodes | 1 |
| **Node details** | |
| Activation function | Unipolar Sigmoid |
| Temperature | 1 |
| **Training** | |
| Mode | Supervised |
| Algorithm | Backpropagation |
| Weight updates | Each epoch |
| Learning rate | 0.01 |
| Momentum rate | 0.95 |

$y_i^W$ as follows,

$$y_i^W = \frac{y_i}{1.50} + f(y_i) \tag{21}$$

where

$$f(y_i) = \begin{cases} \frac{y_i}{1+e^{-0.20(y_i-1.00)}} & y_i \le 1.00 \\ y_i & 1.00 \le y_i \le 10.00 \\ \frac{y_i}{1+e^{-(y_i-10.00)}} & y_i \ge 10.00 \end{cases} \tag{22}$$

Table 4 specifies the two neural networks used in our case study. Since the neural networks use the unipolar sigmoid function as their activation function for all the nodes, the output data was scaled to the range $[0, 1]$. After the training process, the output was converted back to the original scale.

Because the data set in this case study was large, the training process for the two neural networks could be slow. Thus, we normalized the input data as well. We found that the speed of the neural networks was improved after normalizing the input data.

We evaluated the accuracy of the model using the three subsequent releases. The three different types of $AAE$ and $AA^W$ values with $h = 0.05$, and 0.10 for the *fit* data set and three test data sets are given in Table 5.

The column *Hit Rate* indicates how many $y_i$ fall in its $h$-level set $[\widehat{Y}(x)]_h$. Table 5 shows that FNR has a very good hit rate for all the releases in this case study. For example, consider Release 2 for $h = 0.10$. Explanations for the other results are similar. We see that 90.58% of Release 2 modules' actual number of faults

**Table 3. Domain Pattern for Product Metrics**

| | *PCA1* | *PCA2* | *PCA3* | *PCA4* | *PCA5* | *PCA6* |
|---|---|---|---|---|---|---|
| *CALUNQ* | **0.902** | 0.052 | 0.104 | 0.232 | 0.174 | 0.062 |
| *VARUSDUQ* | **0.895** | 0.189 | 0.153 | 0.177 | 0.147 | 0.194 |
| *LOC* | **0.886** | 0.281 | 0.182 | 0.170 | 0.164 | 0.145 |
| *NDSENT* | **0.880** | -0.111 | 0.018 | 0.184 | 0.110 | 0.172 |
| *STMEXE* | **0.869** | 0.259 | 0.176 | 0.173 | 0.269 | 0.072 |
| *STMCTL* | **0.867** | 0.261 | 0.274 | 0.176 | 0.085 | 0.174 |
| *NDSEXT* | **0.847** | 0.020 | 0.109 | 0.201 | 0.086 | 0.353 |
| *STMDEC* | **0.846** | 0.201 | 0.142 | 0.149 | 0.071 | 0.149 |
| *IFTH* | **0.846** | 0.342 | 0.279 | 0.182 | 0.104 | 0.107 |
| *NDSINT* | **0.842** | 0.344 | 0.276 | 0.153 | 0.185 | 0.109 |
| *CNDNOT* | **0.835** | 0.312 | 0.262 | 0.152 | 0.237 | 0.175 |
| *LOP* | **0.828** | 0.108 | 0.208 | 0.017 | 0.021 | -0.096 |
| *VARGLBUS* | **0.802** | 0.360 | 0.201 | 0.144 | 0.212 | 0.205 |
| *VARUSD2* | **0.791** | 0.441 | 0.271 | 0.112 | 0.181 | 0.130 |
| *CAL2* | **0.597** | 0.204 | 0.073 | 0.193 | 0.569 | -0.053 |
| *VARSPNSM* | 0.392 | **0.860** | 0.177 | 0.104 | 0.067 | 0.084 |
| *VARSPNMX* | 0.140 | **0.835** | 0.177 | 0.352 | 0.104 | 0.091 |
| *CNDSPNMX* | 0.121 | 0.276 | **0.757** | 0.143 | 0.256 | 0.306 |
| *CTRNSTMX* | 0.322 | 0.096 | **0.709** | 0.421 | -0.007 | -0.016 |
| *CNDSPNSM* | 0.610 | 0.216 | **0.642** | 0.007 | 0.220 | 0.131 |
| *FILINCUQ* | 0.396 | 0.258 | 0.155 | **0.727** | -0.036 | 0.170 |
| *LGPATH* | 0.210 | 0.380 | 0.358 | **0.640** | 0.170 | -0.042 |
| *KNT* | 0.214 | 0.069 | 0.175 | -0.006 | **0.889** | 0.097 |
| *NDSPND* | 0.402 | 0.149 | 0.217 | 0.075 | 0.084 | **0.816** |
| Variance | 11.616 | 2.821 | 2.372 | 1.696 | 1.643 | 1.230 |
| % Var. | 48.40% | 11.75% | 9.88% | 7.06% | 6.85% | 5.13% |
| Cum. % | 48.40% | 60.15% | 70.03% | 77.09% | 83.94% | 89.07% |

Stopping rule: at least 89% of variance

**Table 5. Accuracy of the Models**

| $h$ | Release | $AAE^{-}$ | $AAE^{C}$ | $AAE^{+}$ | $AA^{W}$ | Hit Rate |
|---|---|---|---|---|---|---|
| 0.10 | 1* | 1.02 | 0.82 | 2.85 | 3.74 | 95.23% |
| | 2 | 0.72 | 0.74 | 1.96 | 2.50 | 90.58% |
| | 3 | 0.79 | 0.85 | 2.02 | 2.60 | 90.54% |
| | 4 | 0.80 | 0.90 | 1.68 | 2.24 | 89.64% |
| 0.05 | 1* | 1.02 | 0.82 | 3.16 | 4.07 | 96.46% |
| | 2 | 0.74 | 0.74 | 2.28 | 2.87 | 92.34% |
| | 3 | 0.80 | 0.85 | 2.43 | 3.06 | 92.66% |
| | 4 | 0.80 | 0.90 | 2.18 | 2.80 | 91.73% |

* Release 1 was *fit* data set

fall within its predicted interval. The average distance between the actual number of faults and the lower limit of the intervals is 0.72 and average distance to the upper limit is 1.96. The average distance between the actual number of faults and the center of the interval is 0.74, and the average width of the interval is 2.50.

An $h$-level set $[\widehat{Y}(\mathbf{x}_i)]_h$ is an interval that indicates to management the likely range of the number of faults in module $i$. If the fault interval for a certain module is close to zero, then the number of faults in this module may be close to zero, and we expect the number of faults discovered in this module to be small. On the other hand we should pay more attention during testing to those modules whose fault intervals are far from zero. A software manager might use this information to allocate testing efforts more effectively.

## 4. Conclusion

High reliability is an important attribute for high-assurance software systems. Consequently, software developers are seeking ways to forecast and improve quality before release. Because many quality factors cannot be measured until after software becomes operational, software quality models are developed to predict quality factors based on measurements that can be collected earlier in the life cycle.

Due to the incomplete information in the early life cycle of software development, a software quality model with fuzzy characteristics can perform better, because fuzzy concepts deal with phenomena that are vague in nature. Fuzzy Nonlinear Regression modeling (FNR) introduced in our study integrates the fuzzy logic and neural networks together; the statistical distributions of the dependent variable are also considered for the model.

A case study of a full-scale industrial software system using FNR found that FNR is a very promising technology for early-stage prediction.

## Acknowledgments

We thank Wendell Jones and the EMERALD team at Nortel Networks for collecting the data sets and Jing Zuo for preparing some of the graphs in our study. This work was supported in part by Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division (Independent Verification and Validation Facility). The findings and opinions in this paper belong solely to the authors and are not necessarily those of the sponsors, or collaborators.

## References

[1] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* PWS Publishing Company, Boston, 1996.

[2] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand. EMERALD: Software metrics and models on the desktop. *IEEE Software,* 13(5):56–60, Sept. 1996.

[3] H. Ishibuchi and H. Tanaka. Interval regression analysis by mixed 0-1 integer programming problem. *Journal of Japan Industrial Management Association,* 40(5):312–319, 1989.

[4] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis.* Prentice Hall, Englewood Cliffs, New Jersey, 3 edition, 1992.

[5] T. M. Khoshgoftaar and E. B. Allen. Multivariate assessment of complex software systems: A comparative study. In *Proceedings of the First International Conference on Engineering of Complex Computer Systems,* pages 389–396, Fort Lauderdale, FL, Nov. 1995. IEEE Computer Society.

[6] T. M. Khoshgoftaar and E. B. Allen. Classification of fault-prone software modules: Prior probabilities, costs, and model evaluation. *Empirical Software Engineering: An International Journal,* 3(3):275–298, Sept. 1998.

[7] T. M. Khoshgoftaar and E. B. Allen. The stability of software quality models over multiple releases. Technical Report TR-CSE-98-25, Florida Atlantic University, Boca Raton, Florida USA, Nov. 1998.

[8] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Accuracy of software quality models over multiple releases. *Annals of Software Engineering,* 6, 2000. In press.

[9] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Early quality prediction: A case study in telecommunications. *IEEE Software,* 13(1):65–71, Jan. 1996.

[10] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. The impact of software evolution and reuse on software quality. *Empirical Software Engineering: An International Journal,* 1(1):31–44, 1996.

[11] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Predictive modeling of software quality for very large telecommunications systems. In *Proceedings of the International Communications Conference,* volume 1, pages 214–219, Dallas, TX, June 1996. IEEE Communications Society.

[12] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning. Application of neural networks for predicting faults. *Annals of Software Engineering,* 1:141–154, 1995.

[13] G. J. Klir. Fuzzy arithmetic with requisite constraints. *Fuzzy Sets and Systems,* 91:165–175, 1997.

[14] C. T. Lin and C. S. G. Lee. *Neural Fuzzy Sytems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice Hall, Inc., Upper Saddle River, New Jersey, 1996.

[15] G. Peter. Fuzzy linear regression with fuzzy intervals. *Fuzzy Sets and Systems,* 63:45–55, 1994.

[16] N. F. Schneidewind. Software metrics validation: Space shuttle flight software example. *Annals of Software Engineering*, 1:287–309, 1995.

[17] L. H. Tsoukalas and R. E. Uhrig. *Fuzzy and Neural Approaches in Engineering*. John Wiley & Sons, Inc., New York, 1997.

[18] Z. Xu, T. M. Khoshgoftaar, and E. B. Allen. Application of fuzzy linear regression model for predicting program faults. In H. Pham and M.-W. Lu, editors, *Proceedings: Sixth ISSAT International Conference on Reliability and Quality in Design*, pages 96–101, Orlando, Florida USA, Aug. 2000. International Society of Science and Applied Technologies.

[19] Z. Xu, T. M. Khoshgoftaar, and E. B. Allen. Early prediction of software quality using fuzzy nonlinear regression model. Technical Report TR-CSE-00-17, Florida Atlantic University, Boca Raton, Florida USA, May 2000.

[20] L. A. Zadeh. Fuzzy sets as a basis for theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

## A. Principal Components Analysis

Software metrics are often highly correlated with one another, because the attributes of software components are often related. Unfortunately, if the independent variables of a model are highly correlated, estimates of the model parameters may not be stable. In other words, insignificant variations in the data may result in drastically different parameter estimates. Principle components analysis (PCA) is a widely used statistical method to reduce the effect of those correlations. Principle components analysis can transform the original set of correlated variables into a smaller set of uncorrelated variables that are linear combinations of the original ones, for more robust modeling [5]. We call the new principal component variables "domain metrics".

Suppose we have $n$ software modules and each has $m$ measurements, let us define an $n \times m$ matrix $\mathbf{Z}$ as the standardized metric data, in which the raw metrics are normalized to mean of zero and standard deviation of one in each of the $n$ rows. The principal components analysis algorithm works as follows [4, 10]:

1. Calculate the covariance matrix, $\mathbf{\Sigma}$, of $\mathbf{Z}$

2. Calculate eigenvalues $\lambda_j$ and eigenvectors $\mathbf{e}_j$ of $\mathbf{\Sigma}, j = 1, \ldots, m$.

3. Reduce the dimensionality of the data. For example, if we choose to explain at least 90% of the total variance of the original standardized metrics, we then choose the minimum $q$ such that $\sum_{j=1}^{q} \frac{\lambda_j}{m} \geq 0.90$.

4. Calculate a standardized transformation matrix, $\mathbf{T}$, where each column is defined as

$$\mathbf{t}_j = \frac{\mathbf{e}_j}{\sqrt{\lambda_j}} \quad \text{for } j = 1, \ldots, q \tag{23}$$

5. Calculate domain metrics for each module, where

$$\mathbf{D}_j = \mathbf{Z}\mathbf{t}_j \tag{24}$$

$$\mathbf{D} = \mathbf{Z}\mathbf{T} \tag{25}$$

The final result $\mathbf{D}$ is an $n \times q$ matrix of domain metrics, where each domain metric, $\mathbf{D}_j$, has a mean of zero and a variance of one.