E-ISSN: 2717-3453 | P-ISSN: 2783-1442

Journal of Fuzzy Extension and Applications

www.journal-fea.com

J. Fuzzy. Ext. Appl. Vol. 4, No. 3 (2023) 188–206.

Paper Type: Research Paper

Interval Type-2 Fuzzy Logic System for Early Software Reliability Prediction

Ini John Umoeka[1,*], Veronica Neekay Akwukwuma[2]

[1] Department of Computer Science, Faculty of Science, University of Uyo, Uyo, Akwa Ibom State, Nigeria; iniumoeka@uniuyo.edu.ng.
[2] Department of Computer Science, Faculty of Physical Sciences, University of Benin, Benin City, Nigeria; nakwukwuma@uniben.edu.ng.

Citation:

Umoeka, I. J., & Akwukwuma, V. N. (2023). Interval type-2 fuzzy logic system for early software reliability prediction. *Journal of fuzzy extension and applications*, 4(3), 188-206.

Received: 02/01/2023 Reviewed: 05/02/2023 Revised: 19/04/2023 Accepted: 11/05/2023

Abstract

The reliability of software product is seen as critical quality factor that cannot be overemphasized. Since real world application is loaded with high amount of uncertainty, such as applicable to software reliability, there should be a technique of dealing with such uncertainty. This paper presents a reliability model to effectively handle uncertainty in software data to enhance reliability prediction of software at the early (requirements and design) stages of Software Development Life Cycle (SDLC). In this paper, a hybrid methodology of Takagi Sugeno Kang (TSK)-based Interval Type-2 Fuzzy Logic System (IT2FLS) with Artificial Neural Network (ANN) learning is employed for the prediction of software reliability. The parameters of the model are optimized using Gradient Descent (GD) back-propagation method. Relevant reliability software requirement and design metrics and software size metrics are utilized as inputs. The proposed approach uses twenty-eight real software project data. The performance of the model is evaluated using five performance metrics and found to provide output values that are very close to the actual output showing better predictive accuracy.

Keywords: Software reliability, Software metrics, Software fault prediction, ANN, Fuzzy logic, Interval type-2 fuzzy Logic system, Gradient descent algorithm.

1 | Introduction

Till date, software has become the fundamental part of various domains of human endeavour. These domains include education, defense, transportation, medical and industries which are directly or indirectly benefiting from software. Diverse historical reports show the impact of software failures experienced round the globe [1]. Such software failures may lead to property, monetary and human losses. Therefore, the reliability of software cannot be ignored. The reliability of software at the early phase of software life cycle is very important [2]. Hence, it is necessary to guarantee the reliability of software systems by fixing the faults early during software development process and this will significantly minimize the cost of testing in later stages. Software reliability and quality has become a fundamental focus for both designers and developers during software development process. Software reliability is the probability to perform failure free operation and produce correct output for a specified time under specified conditions [3], while software quality is defined as the degree to which a system

Corresponding Author: iniumoeka@uniuyo.edu.ng

https://doi.org/10.22105/jfea.2023.379363.1241

component or process meets customer requirement [4]. According to Arasteh [5], reliable software is a software that is void of defects and one approach of enhancing reliability of software is predicting defects before testing phase. Since reliability has become a principal factor in software systems, its prediction is very paramount. Most software are difficult and complicated in nature, thus making its reliability analyses and predictions also difficult and complicated. This makes it difficult for software developers to determine the level of reliability. Many Software Reliability Prediction (SRP) models exist in literature; however, practitioners find it very hard to employ these methods in software development process since they have to decide on the amount of data to collect and suitable software reliability model and techniques to implement. Since the nature of software engineering involves making of measurements, reliability prediction methods continue to be identified so as to assist software developers; and in order to express software product reliability, reliability relevant metrics are needed.

A general technique to measure the quality of software product is to identify the presence of defects in it, and usually the metric used for it is software reliability. As opined by Kaur and Sharma [6] that all software has the propensity of containing defects (product anomaly) since it is very difficult to develop fault free software product [7]. Software defect prediction is a method of building machine learning classifiers to predict defective code snippets using historical information in software repositories [8]. The predicted results can therefore aid developers in locating and fixing possible defects thereby enhancing the reliability of software [9]. Software defect prediction therefore, is an important feature of preventive maintenance, which involves applying various computing methods to predict potential defects in software product before the software fails [8]. Software maintenance is a process that continuously takes place throughout the entire activity of software development so as to guarantee the performance of the software as expected.

SRP can take place in the later or in the early phase of Software Development Life Cycle (SDLC) [10]–[15]. The accuracy of software reliability using software reliability models is mostly possible at the later stage of [13]. However, the prediction of reliability in the early phase of SDLC is cost effective [13]. Most of SRP models depend on failure or fault data for prediction. But, in the early phases of SDLC, failure or fault data are unavailable which forms a major challenge. However, in the early phases of SDLC, qualitative values of software metrics are available and this could be employed in predicting residual defects in software. The metrics which have impact on software reliability in SDLC are indicated in [16]–[19]. In fact, most of these software metrics have high amount of uncertainty resulting either from unrealistic assumptions and some measures that cannot be specifically defined in the process of software development. According to Debnath [20], uncertainty is everyday, everywhere and in different forms, there should therefore be a technique of dealing with the uncertainty. Hence, the idea of Fuzzy Logic System (FLS), that forms a fulcrum in uncertainty modeling, is of utmost importance

In order to guarantee the development of quality software, various software defects prediction models have presented based on different methods including software metrics, statistical approaches, machine learning approaches, classification methods and other traditional approaches. For instance, Khoshgoftaar et al. [21] presented a neural network for predicting the number of faults and introduced an approach for static reliability modeling. The authors trained two neural networks; one with the complete set of principal components and the other one with the set of components selected by multiple regression model selection. Comparison of their models showed a better understanding of neural network software quality models. Karunanithi [22], [23] designed neural network based software reliability model to predict cumulative number of failures. Their model used feed-forward neural network, recurrent neural network and Elman neural network and also used execution time as the input of the network. Tian and Noore [24] and Tian and Noore [25] proposed a genetic algorithm to optimize the number of delayed input neurons and the number of neurons in the hidden layer of the neural network for the prediction of software reliability and software failure time. Oliveira et al. [26] used boosting techniques to improve software reliability models based on genetic programming. Boosting technique combines several hypotheses of the training set to get better results. Sharma et al. [27] investigated the applicability of the modified artificial bee colony algorithm to estimate the parameters of Software Reliability Growth Models (SRGMs). The estimated model parameters were then used to predict the faults in a software system during the testing process.

Khoshgoftaar et al. [28] presented a neural network model to predict software quality using large telecommunication system to classify modules as fault prone or not fault-prone; Khoshgoftaar et al. [29] made use of regression trees with classification rule to classify fault-prone software modules using data from a very large telecommunications system. In [30], genetic algorithm is applied for early detection of defect in software modules. Kanmani et al. [31] predicted software defect in object-oriented software using neural network. In [32] different techniques have employed in the prediction of software fault which include soft computing, data mining and machine learning methods with their accuracy rates based on Random Forest, Decision Tree Regression, Neural Network, Genetic Algorithm, SVM, Artificial Neural Network (ANN) and fuzzy logic. Defect Prediction framework using Attention-based Recurrent Neural Network (DP-ARNN) is presented in [9]. This model automatically learned syntactic and semantic features for accurate defect prediction. The framework employed seven Java projects in Apache for its validation using F1 measure and Area Under Curve (AUC) as evaluation criteria.

Many other research works on software fault prediction have been presented in the literature using alternative modeling and ensemble approaches. For example, Singh et al. [7] proposed an integrated system with ability of finding functional software metrics (features) and concurrently producing a set of human interpretable fuzzy rules for software fault prediction. Erturk and Sezer [33] presented a model to predict software fault based on fuzzy logic and ANN. Arasteh [5] presented a hybrid prediction model of neural network and naïve bayes algorithm to build software fault prediction model. This model used five traditional fault datasets to develop and assess the prediction model. According to Adark [34], all these authors applied various performance measures; including Matthews Correlation Coefficient (MCC), precision, recall, Receiver Operating Characteristic (ROC), Root Mean Squared Error (RMSE), ROC-AUC, Accuracy, Balance, Geometric Mean (GM), showing that defection prediction procedure does not have a standard approach of assessing the proposed models. Fenton et al. [10] presented a causal model using Bayesian Nets for early life cycle defect prediction. Their model predicted the number of residual defects that are likely to be found during independent testing or operational usage. Their model was used to analyze several evaluation measures on a dataset, extracted from 31 completed software projects in the consumer electronics industry, which was obtained by means of a questionnaire distributed to managers of projects. Their model validation also authenticated the necessity for using the qualitative data in the model. Nevertheless, these models proposed for early software defects prediction are insufficient to provide reliable results because of presence of vague and imprecise data.

The fuzzy set theory presents a method of dealing with uncertainty, vagueness and imprecision found in data. Hence the use of fuzzy logic is more suitable for the prediction of software defects in early phase of software development process, where failure data is unavailable. Some research works have been proposed based on FLSs to predict software defects because of the presence of vagueness or fuzziness in software data. For instance, [12]–[17] and [33]–[38] made use of Type-1 Fuzzy Logic Systems (T1FLSs) for early software defects prediction. Although T1FLSs have been widely used, literatures show that T1FLSs only handle uncertainty to some degree in many applications and may not reduce their effects in some real world applications [39]. T1FLSs also use Membership Functions (MFs) that are precise, thus making it inadequate to deal with uncertainty associated with the inputs and outputs of FLS [40]–[42]. Also, the uncertainty in Type-1 Fuzzy Sets (T1FSs) disappears the moment its MFs are specified [43], leaving crisp numerical values. Because of this, a higher fuzzy set(s), known as Type-2 Fuzzy Set (T2FS) was introduced by Zadeh [44] as an extension of T1FS having T1FSs as MFSs with third dimension. T2FSs can be General (GTFSs) or Interval (IT2FSs). GT2FS representation is three dimensional with different weights on the third dimension. Working with GT2FS is computationally expensive because its representation. Thus, many researchers prefer the use of IT2FS since its third dimension takes the value of 1 and does not carry any information [45] and can easily be represented on a 2-D plane. These make the use of IT2FS computationally effective. An Interval Type-2 Fuzzy Logic System (IT2FLS) uses T2FSs, that are upper order fuzzy sets with the capacity to sufficiently deal with the uncertainty in the linguistic information [41]–[43]. As stated in Chatterjee et al. [46], type-2 fuzzy systems handle uncertainties that type-1 lacks the ability to handle because their membership grades are fuzzy and can be used to reduce uncertainties in the vague linguistic values of software attributes. Hence, this paper presents an IT2–Takagi Sugeno Kang (TSK)-FLS-based software defects prediction with ANN learning at the early phases (requirement and design) of SDLC.

Chatterjee et al. [46] developed a fuzzy-rule based generation algorithm using Mamdani fuzzy inference IT2FLS and Gaussian MFs with uncertain mean to predict fault in early phase of software development. Although quite a number of research works have been proposed using T1FLS; [46] applied IT2FLS to present a model in the area of early software fault prediction. Sadly, the accuracies of the developed models are not adequate, therefore more works need to be done particularly as there are emergence of new modeling methods sporadically being developed, taking cognizance of the weaknesses of the previous works [47]. As far as software engineering is concerned, various methods to software defect prediction are still emerging as potential fields of research [48].

This paper therefore, presents for the first time an enhanced early software defect prediction at the requirement and design phases of SDLC based on IT2 FLS with optimized parameters using TSK fuzzy inference. The intention is to minimize uncertainty thereby increasing the performance of the IT2FLS software defects prediction in terms of accuracy with minimal computational cost. These (optimization and TSK inferencing) present the novelty of this work with respect to early software defects prediction. The proposed IT2FLS-SRP makes use of Gaussian MF with uncertain standard deviations. As stated in Kayacan and Khanasar [49] the Gaussian MF with uncertain standard deviations is the only known MF that is continuous at all points and well suited for optimization problem undertaken in this work.

1.1 | Main Contributios

The contributions of this paper therefore are as follows: 1) optimization of the parameters of IT2FLS-SRP for the first time using Gradient Descent (GD) back-propagation algorithm, and 2) managing the varying degrees of uncertainties in the rule base using a user-defined parameter, β in the requirements and design phases of early SRP.

1.2 | Paper Organization

The rest of this paper is arranged as follows: Section 2 shows the preliminaries presenting definition of terms. Section 3 describes the methodology. The parameter update rule is presented in Section 4. Section 5 has the experimental set while Section 6 describes the performance evaluation and Section 7 discusses the findings and conclusion.

2 | Preliminaries

An IT2FS is characterized by a FOU defined by a Lower Membership Function (LMF), $\underline{\mu}_{\tilde{A}}(x,u)$ and an Upper Membership Function (UMF) $\overline{\mu}_{\tilde{A}}(x,u)$ for all $x \in X$ where $\underline{\mu}_{\tilde{A}}(x,u) = 1$ and $\overline{\mu}_{\tilde{A}}(x,u) = 1$. Hence, the IT2FS can also be represented as:

$$\widetilde{\mathrm{A}} = \{((\mathrm{x},\mathrm{u}),\underline{\mu}_{\widetilde{\mathrm{A}}}(\mathrm{x},\mathrm{u}),\overline{\mu}_{\widetilde{\mathrm{A}}}(\mathrm{x},\mathrm{u})) \mid \text{for all } \mathrm{x} \in \mathrm{X}, \text{for all } \mathrm{u} \in \mathrm{J}_{\mathrm{x}}.$$

$$\widetilde{\mathrm{A}} = \int_{\mathrm{x}\in\mathrm{X}}\int_{\mathrm{u}\in\mathrm{J}_{\mathrm{x}}} 1/(\mathrm{x},\mathrm{u})\ \ \mathrm{J}_{\mathrm{x}} \in [0,1].$$

$$\widetilde{\mathrm{A}} = \left\{\left((\mathrm{x},\mathrm{u}),\underline{\mu}_{\widetilde{\mathrm{A}}}(\mathrm{x},\mathrm{u}),\overline{\mu}_{\widetilde{\mathrm{A}}}(\mathrm{x},\mathrm{u})\right) \mid \text{for all } \mathrm{x} \in \mathrm{X}, \text{for all } \mathrm{u} \in \mathrm{J}_{\mathrm{x}} \subseteq [0,1]\right\},$$

Or

$$\widetilde{\mathrm{A}} = \sum_{\mathrm{x}\in\mathrm{X}}\sum_{\mathrm{u}\in\mathrm{J}_{\mathrm{x}}} 1/(\mathrm{x},\mathrm{u})\,\mathrm{J}_{\mathrm{x}} \in [0,1],$$

where and $\int$ and $\sum$ represent the union of all admissible points in a continuous and discrete Universe of Discourse (UoD) respectively [45]. A finite UoD is assumed in this study. In this research, the IT2FS is defined using interval type-2 Gaussian function with a fixed mean and uncertain standard deviation as shown in *Fig. 1*.

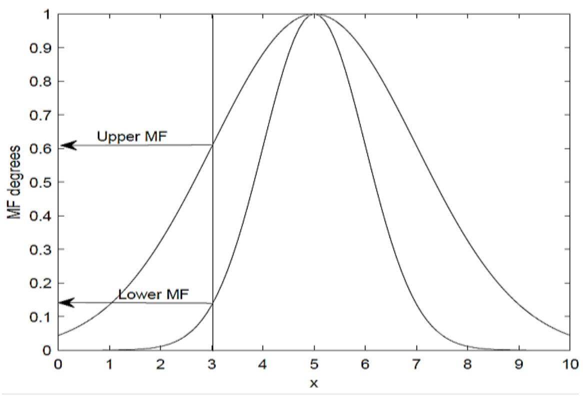


Fig. 1. A Gaussian interval T2FS.

Thus, for IT2FSs, the FOUs are generated namely LMF and UMF FOUs defined as in *Eq. (1)*.

$$FOU_{\mu}\left(\tilde{A}\right) = \cup_{\forall \in X} \left[\underline{\mu}_{\tilde{A}}(x), \overline{\mu}_{\tilde{A}}(x)\right]. \quad (1)$$

The FLS that makes use of IT2FSs is called an and is shown in *Fig. 2*.

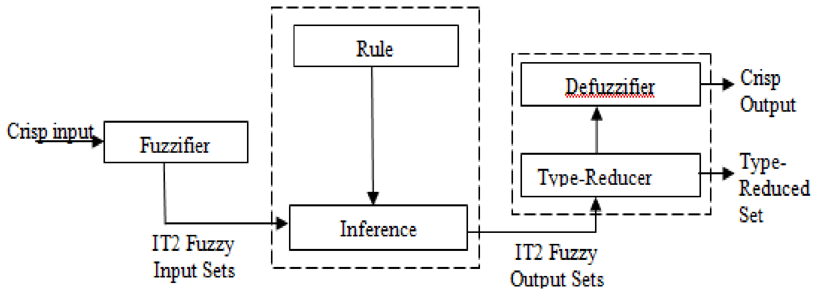


Fig. 2. Architecture of the proposed IT2FLS for software defects prediction.

3 | Proposed Model for Early Software Defects Prediction

The proposed model predicts number of defects in the early phases (requirements and design) of SDLC based on interval type-2 TSK-Fuzzy Logic System (IT2-TSK-FLS). The model prediction follows from the fact that the reliability of software is dependent on the number of defects in the software product [50]. Inputs for the model are Relevant-Reliability Software Metrics (RRSMs) of each phase and software size (thousand lines of codes-KLOC) which are fed into the IT2-TSK-FLS for processing and generation of the model outputs. The output for the model is the number of defects predicted at the end of requirements phase known as Requirements Phase Number of Defects (RPND) which serves as one of the inputs to the design phase whose output is called Design Phase Number of Defects (DPND). The proposed model architecture is shown in *Fig. 3*.

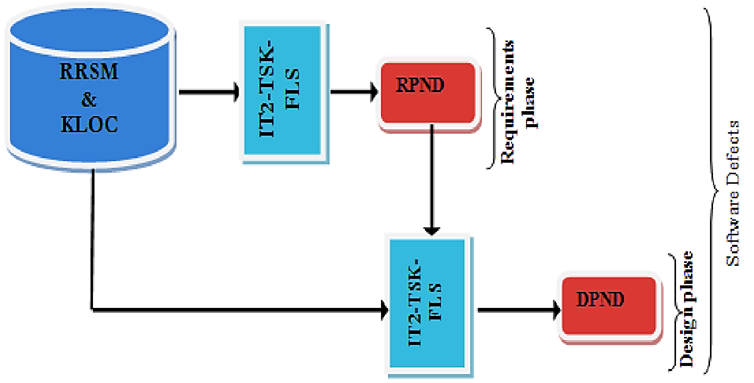


Fig. 3. Proposed model architecture.

The proposed model made use of the software metrics in subsections 3.1 and 3.2 respectively.

3.1 | Software Metrics for Requirements Phase

I. Requirement Complexity (RC): An increase in RC increases the number of defects, thereby decreasing the reliability of the software at the requirement phase [2], [46], [51].

II. Requirement Stability (RS): Low RS which can result in increased number of defects in software and that a higher value of RS, results in higher reliability [2].

III. Review, Inspection and Walkthrough (RIW): A high level of RIW results in more defects identified and removed leaving fewer defects in the requirement phase [2], [15].

3.2 | Software Metrics for Design Phase

I. Design Team Experience (DTE): More of this metric means lesser number of defects in the design stage resulting in high reliability.

II. Process Maturity (PM): Increasing the PM, results in lesser number of defects, thereby improving the reliability of software. The impact of this metric on software defect is ranked as [10], [16], [46].

4 | Model Implementation

The IT2-TSK-FLS is implemented using the architectural design model as in *Fig. 4*.

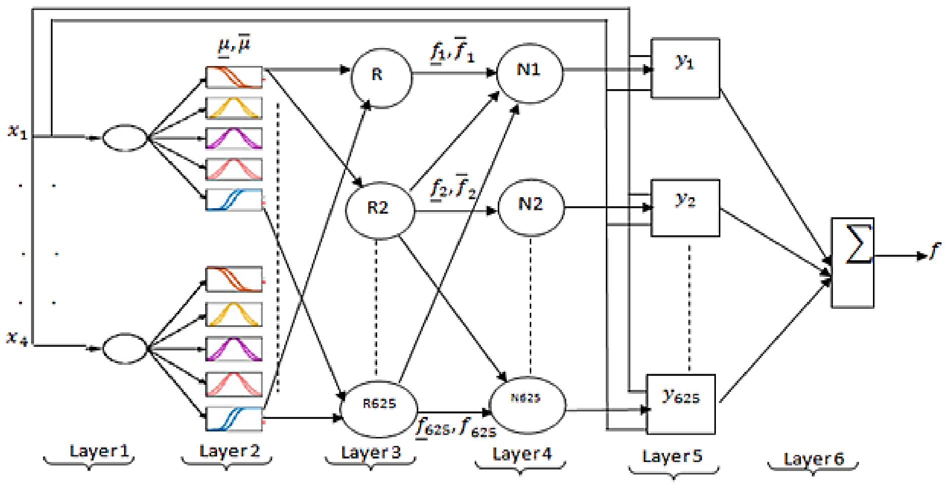

$\underline{\mu}, \bar{\mu}$
x_1
x_4
R
R2
R625
$\underline{f}_1, \bar{f}_1$
$\underline{f}_2, \bar{f}_2$
$\underline{f}_{625}, \bar{f}_{625}$
N1
N2
N625
y_1
y_2
y_{625}
$\sum$
f
Layer1
Layer2
Layer3
Layer4
Layer5
Layer6

Fig. 4. Design specification of TSK-IT2FLS for the model [52].

The architectural design consists of six layers. Input layer (Layer 1): This is where the external input signals are distributed. Membership function layer (Layer 2): translates the external inputs into IT2FSs. IT2FS

(LMF and UMF) are obtained using IT2 Gaussian MF with a fixed mean and uncertain standard deviation and is computed as in [2] by a process called fuzzification and are indicated in *Eqs. (2)* and *(3)*.

$$\underline{\mu}_{\tilde{A}_{ik}}(x_i) = exp\left[-\frac{(x_i - m_{ik})^2}{2\sigma_{1,ik}^2}\right]. \tag{2}$$

$$\overline{\mu}_{\tilde{A}_{ik}}(x_i) = exp\left[-\frac{(x_i - m_{ik})^2}{2\sigma_{2,ik}^2}\right]. \tag{3}$$

$x\prime_i s$ are the input vectors while the parameters, m, σ_1, σ_2 are the antecedents parameters for MFs of IT2FSs.

In this paper, the UoD (range) of the membership grades of all input metrics (linguistic variables) is presented in a normalized form to lie between 0 and 1. The input metrics, with their ranges and linguistic terms, Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH) for the two phases are as shown in *Tables 1* and *2*.

Table 1. Requirements phase metrics with their linguistic terms and fuzzy range.

Input Metrics (Linguistic Variable)	Linguistic Terms	Fuzzy Range
RC	{VL, L, M, H, VH}	{0, 1}
RS	{VL, L, M, H, VH}	{0, 1}
RIW	{VL, L, M, H, VH}	{0, 1}
KLOC	{VL, L, M, H, VH}	{0, 1}

Table 2. Requirements phase metrics with their linguistic terms and fuzzy range.

Input Metrics (Linguistic Variable)	Linguistic Terms	Fuzzy Range
DTE	{VL, L, M, H, VH}	{0, 1}
PM	{VL, L, M, H, VH}	{0, 1}
KLOC	{VL, L, M, H, VH}	{0, 1}
RPTND	{VL, L, M, H, VH}	{0, 1}

The IT2FLS MF graphical representation for all the software input metrics that are partitioned into five linguistic terms is depicted in *Fig. 5*, for the purpose of visualization.

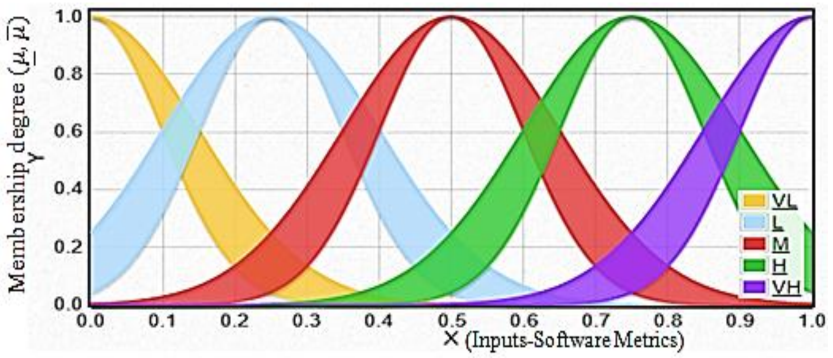


Fig. 5. IT2 Gaussian membership functions for requirement and design metrics input variables.

The parameters of the functions mean and standard deviations for each of the MFs were obtained as follows:

The mean or centre of IT2 Gaussian MFs, the data is partitioned into five sets. The centre of each set is the mean value of each MF. For instance, the mean of VL is 0, L is 0.25, M is 0.5, H is 0.75 and VH is 1. The standard deviations (width of the MFs) are assigned arbitrarily as 0.3 for UMF and 0.2 for LMF.

In layer 3 (Rule layer), the inputs are combined to form fuzzy rules in the rule base. *Eq. (4)* shows the IF-THEN IT2-TKS fuzzy rule formation adopted in this study.

$$R_k: \mathrm{IF} x_1 \mathrm{is} \tilde{A}_{1k} \mathrm{AND} x_2 \text{ is } \tilde{A}_{2k} \text{ AND} \dots \text{ AND } x_n \text{ is } \tilde{A}_{nk} \text{ THEN } y_k = w_{1k}x_1 + w_{2k}x_2 + w_{nk}x_n + b_{k,} \tag{4}$$

where $\tilde{A}_{ik}$ are IT2FSs $(i = 1,2,..,n), y_k(k = 1,2,\dots,K)$ are the output of the k_{th} rule, $w_{ik}$and b_k are the consequents parameters.

This study employs a TSK-based inferencing known as A2-C0 where the antecedent part (A2) is IT2FSs and the consequent (C0) parts are crisp values. The total number of rules used is obtained using the formula.

Number of fuzzy rules = P^I, where *P* is the number of MFs, *I* is the number of input variables. In this study, *I* is 4 and *P* is 5 giving a total number of 625 rules for the rule base in each of the phases. The subset of the fuzzy rules used in this study is presented in *Table 3*.

From the proposed model architecture in *Fig. 3*, the two phases are clearly indicated. Therefore, assigning rules for the metrics in each of the phases is done one after the other.

Table 3. Fuzzy rules for requirement and design input metrics.

Rule No.	Fuzzy Rules for Requirement Phase				Fuzzy Rules for Rules for Design Phase			
	RC	RS	RIW	KLOC	RPND	DTE	PM	KLOC
1	VL	VL	VL	VL	VL	VL	VL	VL
2	VL	VL	VL	L	VL	VL	VL	L
3	VL	VL	VL	M	VL	VL	VL	M
4	VL	VL	VL	H	VL	VL	VL	H
5	VL	VL	VL	VH	VL	VL	VL	VH

For each range of input variables, a set of firing strength are calculated using algebraic product (t-norm operator) operation denoted as * and defined as in *Eqs. (5)* and *(6)*.

$$\underline{f}_k = \prod_{i=1}^{n} \underline{\mu}_{\tilde{A}_{ik}} = \underline{f}_k(x) = \underline{\mu}_{\tilde{A}_{1k}}(x_1) * \underline{\mu}_{\tilde{A}_{2k}}(x_2) * \dots * \underline{\mu}_{\tilde{A}_{nk}}(x_n). \tag{5}$$

$$\bar{f}_k = \prod_{i=1}^{n} \bar{\mu}_{\tilde{A}_{ik}} = \bar{f}_k(x) = \bar{\mu}_{\tilde{A}_{1k}}(x_1) * \bar{\mu}_{\tilde{A}_{2k}}(x_2) * \dots * \bar{\mu}_{\tilde{A}_{nk}}(x_n), \tag{6}$$

where $\underline{f}_k$ and $\bar{f}_k$are firing strengths for lower and upper MFs of the rules respectively.

In layer 4 (normalization layer) the output of layer 3 are normalized to have normalized outputs $(N)_k$using *Eqs. (7)* and *(8):*

$$\underline{N}_k = \frac{\underline{f}_k}{\sum_{k=1}^{M} \underline{f}_k}. \tag{7}$$

$$\bar{N}_k = \frac{\bar{f}_k}{\sum_{k=1}^{M} \bar{f}_k}, \tag{8}$$

where $\underline{N}_k$ and $\bar{N}_k$ are lower and upper firing strengths normalized outputs.

In layer 5 (consequent layer) outputs are obtained using *Eq. (9)*.

$$y_k \mathrm{TSK} = \sum_{i=1} w_{ik} x_i + b_k. \tag{9}$$

Layer 6 (summation layer) gives the final crisp output *(f)* of the model. Existing works in the literature employ Karnik-Mendel iterative type-reduction procedure to reduce a T2FS to (T1FS) which is very tedious and has high computational cost [53]. In the literature various approaches have been developed to directly compute the output of a T2FL thereby circumventing the time wastage type-reduction procedure [54]–[56]. In this study, the Bergian-Melek-Mendel (BMM) algorithm [54] is used to directly compute the IT2FLS outputs as a weighted average of the crisp rule consequents as expressed in *Eq (10)*.

$$y = (1-\alpha) * \frac{\sum_{k=1}^{M} \underline{f}_k y_k}{\sum_{k=1}^{M} \underline{f}_k} + \alpha * \frac{\sum_{k=1}^{M} \bar{f}_k y_k}{\sum_{k=1}^{M} \bar{f}_k}. \quad (10)$$

The parameter α is user defined parameter that weighs the contributions of LMF and UMF to the final output and lies within the range of 0 and 1. If $\alpha = 0$, the LMF alone contributes to the final output and if $\alpha = 1$, UMF only contributes to the final output. If $0 \leq \alpha \leq 1$, then contribution is from both the LMF and UMF.

5 | Parameter Update

In this study, the antecedent and consequent parameters of the IT2FLS are updated using GD back-propagation method so as to reduce the cost function. *Eq. (11)* shows the cost function for a single output:

$$E = \frac{1}{2}\,(y^{a} - y)^2, \quad (11)$$

where y^a is the actual output and y is the expected output of the proposed model. The GD update rules are represented as shown in *Eq. (12)* to *(16)*:

$$w_{ik}(t+1) = w_{ik}(t) - \varphi \frac{\partial E}{\partial w_{ik}}. \quad (12)$$

$$b_k(t+1) = b_k(t) - \varphi \frac{\partial E}{\partial b_k}. \quad (13)$$

$$m_{ik}(t+1) = m_{ik}(t) - \varphi \frac{\partial E}{\partial m_{ik}}. \quad (14)$$

$$\sigma_{1,ik}(t+1) = \sigma_{1,ik}(t) - \varphi \frac{\partial E}{\partial \sigma_{1,ik}}. \quad (15)$$

$$\sigma_{2,ik}(t+1) = \sigma_{2,ik}(t) - \varphi \frac{\partial E}{\partial \sigma_{2,ik}}, \quad (16)$$

where w and b are the consequent parameters, m, $\sigma 1$, and $\sigma 2$ are the antecedent parameters while φ is the learning rate (step size) that must be carefully chosen as a large value may lead to instability, and small value on the other hand may lead to a slow learning process. For this work, the learning rate chosen provided stable learning as shown in the smooth convergence of the error (*Figs. 10* and *11*).

The consequent parameters of IT2FLS are adjusted as in *Eqs. (17)* and *(18)*:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial y_k}\frac{\partial y_k}{\partial w_{ik}} = (y(t) - y^{a}(t)) * \left[\left((1-\alpha) * \frac{\underline{f}_k}{\sum_{k=1}^{M} \underline{f}_k} + \alpha * \frac{\bar{f}_k}{\sum_{k=1}^{M} \bar{f}_k}\right)\right] * x_i. \quad (17)$$

$$\frac{\partial E}{\partial b_k} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial y_k}\frac{\partial y_k}{\partial b_k} = = \left(y(t) - y^{a}(t)\right) * \left[\left((1-\alpha) * \frac{\underline{f}_k}{\sum_{k=1}^{M} \underline{f}_k} + \alpha * \frac{\bar{f}_k}{\sum_{k=1}^{M} \bar{f}_k}\right)\right] * 1, \quad (18)$$

where y_k is the output of the kth rule. The adjustments of the antecedent parameters are shown in *Eqs. (19)* to *(28)*:

$$\frac{\partial E}{\partial m_{ik}} = \frac{\partial E}{\partial y}\left[\frac{\partial y}{\partial \underline{f}_k}\frac{\partial \underline{f}_k}{\partial \underline{\mu}_{ik}}\frac{\partial \underline{\mu}_{ik}}{\partial m_{ik}} + \frac{\partial y}{\partial \bar{f}_k}\frac{\partial \bar{f}_k}{\partial \bar{\mu}_{ik}}\frac{\partial \bar{\mu}_{ik}}{\partial m_{ik}}\right]. \quad (19)$$

$$\frac{\partial E}{\partial \sigma_{1,ik}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial \underline{f}_k}\frac{\partial \underline{f}_k}{\partial \underline{\mu}_{ik}}\frac{\partial \underline{\mu}_{ik}}{\partial \sigma_{1,ik}}. \quad (20)$$

$$\frac{\partial E}{\partial \sigma_{2,ik}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial \bar{f}_k}\frac{\partial \bar{f}_k}{\partial \bar{\mu}_{ik}}\frac{\partial \bar{\mu}_{ik}}{\partial \sigma_{2,ik}}, \quad (21)$$

where

$$\frac{\partial E}{\partial y} = y(t) - y^{a}(t), \frac{\partial y}{\partial \underline{f}_k} = (1-\alpha) * \frac{y_k - \underline{y}}{\sum_{k=1}^{n} \underline{f}_k}, \qquad \underline{y} = \frac{\sum_{k=1}^{n} \underline{f}_k y_k}{\sum_{k=1}^{n} \underline{f}_k}. \tag{22}$$

$$\frac{\partial y}{\partial \bar{f}_k} = \alpha * \frac{y_k - \bar{y}}{\sum_{k=1}^{n} \bar{f}_k}, \qquad \bar{y} = \frac{\sum_{k=1}^{n} \bar{f}_k y_k}{\sum_{k=1}^{n} \bar{f}_k}. \tag{23}$$

$$\frac{\partial \underline{\mu}_k(x_i)}{\partial m_{ik}} = (x_i - m_{ik}) * \frac{\exp\left(-\frac{(x_i - m_{ik})^2}{2 * \sigma_{1,ik}^2}\right)}{\sigma_{1,ik}^2}. \tag{24}$$

$$\frac{\partial \bar{\mu}_k(x_i)}{\partial m_{ik}} = (x_i - m_{ik}) * \frac{\exp\left(-\frac{(x_i - m_{ik})^2}{2 * \sigma_{2,ik}^2}\right)}{\sigma_{2,ik}^2}. \tag{25}$$

$$\frac{\partial \underline{\mu}_k(x_i)}{\partial \sigma_{1,ik}} = (x_i - m_{ik})^2 * \frac{\exp\left(-\frac{(x_i - m_{ik})^2}{2 * \sigma_{1,ik}^2}\right)}{\sigma_{1,ik}^3}. \tag{26}$$

$$\frac{\partial \underline{\mu}_k(x_i)}{\partial \sigma_{2,ik}} = (x_i - m_{ik})^2 * \frac{\exp\left(-\frac{(x_i - m_{ik})^2}{2 * \sigma_{2,ik}^2}\right)}{\sigma_{2,ik}^3}. \tag{27}$$

Using the product t-norm (*) operation, then,

$$\frac{\partial \underline{f}_k}{\partial \underline{\mu}_{ik}} = \prod_{\substack{p=1 \\ p \neq i}}^{N1} \underline{\mu}_{pk}, \frac{\partial \bar{f}_k}{\partial \bar{\mu}_{ik}} = \prod_{\substack{p=1 \\ p \neq i}}^{N1} \bar{\mu}_{pk}. \tag{28}$$

The value of α is adjusted using *Eq. (29)*.

$$\alpha(t+1) = \alpha(t) - \varphi \frac{\partial E}{\partial \alpha}, \qquad \frac{\partial E}{\partial \alpha} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial \alpha} = (y - y^{a})\left(\frac{\underline{f}_k * y_k}{\sum_{k=1}^{M} \underline{f}_k} + \frac{\bar{f}_k * y_k}{\sum_{k=1}^{M} \bar{f}_k}\right). \tag{29}$$

6 | Model Implementation

The proposed model is implemented using the project datasets in PROMISE repository [57]. The software defects dataset in [56] is a benchmark dataset and has been adopted in many literature [10]–[13], [46], [50] made publicly available to promote repeatable, verifiable, refutable and/or improvable predictive models of software engineering. The datasets contain twenty eight real software projects obtained from an electronics consumer's company as reproduced in *Table 4*:

Table 4. Twenty-eight software project with metric inputs.

SN	Project Number	RC	RS	RIW	DTE	DPF	SIZE (KLOC)	Actual Defect
1	1	M	L	VH	L	H	6	148
2	2	L	H	VH	L	H	0.9	31
3	3	H	H	VH	H	H	53.9	209
4	5	H	M	H	L	H	14	373
5	6	M	H	VH	M	M	14	167
6	7	L	M	VH	M	H	21	204
7	8	M	H	H	H	M	5.8	53
8	9	L	H	VH	VH	VH	2.5	17
9	10	M	H	H	H	H	4.8	29
10	11	H	H	H	H	H	4.4	71
11	12	H	L	H	VH	M	19	90
12	13	H	L	M	H	H	49.1	129
13	14	VH	H	H	H	H	58.3	672
14	15	H	VL	H	H	H	54	1768
15	16	L	M	H	H	H	26.7	109

Table 4. Continued.

SN	Project Number	RC	RS	RIW	DTE	DPF	SIZE (KLOC)	Actual Defect
16	17	L	M	M	M	H	33	688
17	18	VH	VL	H	M	H	155.2	1906
18	19	H	M	H	H	H	87	476
19	20	VH	VL	M	VL	L	50	928
20	21	L	M	H	H	H	22	196
21	22	M	L	M	H	L	44	184
22	23	H	M	VH	L	H	61	680
23	24	M	L	M	M	H	99	1597
24	27	H	M	VH	M	M	52	412
25	28	VH	L	VH	M	M	36	881
26	29	M	VH	VH	VH	H	11	95
27	30	L	VH	VH	H	H	1	5
28	31	M	M	H	H	H	33	653

The inputs into the system comprise software metrics of requirements and design phases of SDLC with software size. In designing the system, five MFs (VL, L, M, H, and VH) are utilized. The experiment is run on MATLAB© 2015a.

The number of parameters in the network to be optimized includes: $m = n * d; \sigma_1 = n * d; \sigma_2 = n * d; w = n * M; b = 1 * M$.

Thus, the total number of parameters to be optimized in each phase is $3(n * d) + M(n + 1)$, where n = number of inputs, d = number of MF and M is the number of rules.

Hence, the total number of parameters to be optimized in the proposed model is $3(4 * 5) + 625(4 + 1) = 3185$ parameters. During learning the initial values of consequent parameters, w and b were randomly selected in the interval [0, 1]. The optimized IT2FLS-based model is executed for 100 epochs in each phase with learning rate, $\varphi = 0.8$, for both requirements and design phases respectively. The parameter, α, value is initialized to 0.5 to guarantee equal initial lower and upper membership contributions to the final output.

7 | Performance Evaluation

To determine the performance accuracy of the proposed model, the results of the proposed model are compared with the results in the previous works in the literature adopting five performance metrics in [45] model as shown in *Eqs. (30)* to *(34)*:

$$\text{Normalized Root Mean Square Error (NRMSE)} = \frac{\text{RMSE}}{y_{\max}} - y_{\min}. \tag{30}$$

$$\text{Mean Magnitude of Relative Error(MMRE): MMRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i^a - y_i|}{y_i}. \tag{31}$$

$$\text{Balanced Mean Magnitude of Relative Error(BMMRE)} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i^a - y_i|}{\text{Min}(y_i^a, y_i)}. \tag{32}$$

Smaller values of RMSE, NRMSE, MMRE and BMMRE denote better prediction accuracy [13], [45].

$$\text{Coefficient of Determination}\left(R^2\right) = 1 - \frac{\sum_{i=1}^{n}(y_i^a - y_i)^2}{\sum_{i=1}^{n}(y_i^a - \bar{y})^2}, \tag{33}$$

where $\bar{y}$ is the mean of the actual values, y_i^a. Better prediction accuracy is achieved as result of R^2 is closer to 1. In *Eqs. (29)* to *(34)*, y_i^a is the actual value, $y_i$is the predicted value and n is the number of testing data points. The metrics in *Eqs. (29)* to *(34)* are chosen because they give better prediction accuracy. For effective learning, the KLOC values are normalized to lie between 0 and 1. After prediction, the final results are de-normalized to obtain the predicted values in their original states. The predicted result of the proposed study is compared with [45] as shown in *Table 5*.

Table 5. Predicted no. of defects at requirements and designphases.

SN	Project No.	Actual No. of Defect	Predicted No. of Defects Chatterjee et al. [46]		Predicted No. of Defects Proposed Model	
			Requirements Phase	Design Phase	Requirements Phase	Design Phase
1	1	148	85	72	145.9	148.8
2	2	31	37	12	29	116.3
3	3	209	-	-	157.6	246.9
4	5	373	-	-	308.4	451.8
5	6	167	-	-	164.3	217
6	7	204	139	168	299.1	249.7
7	8	53	-	-	55.8	59.1
8	9	17	41	18	62.3	-8.3
9	10	29	64	43	-21	72.3
10	11	71	-	-	79.9	57.5
11	12	90	219	155	73.2	111.3
12	13	129	-	-	174.6	134.5
13	14	672	-	-	680.5	758.3
14	15	1768	1946	1549	1766.3	1764.5
15	16	109	-	-	138	166.4
16	17	688	371	286	762	701.7
17	18	1906	-	-	1905.7	1905.4
18	19	476	-	-	472.3	483.2
19	20	928	-	-	913.7	930.2
20	21	196	-	-	193.5	319.8
21	22	184	-	-	199	196.1
22	23	680	-	-	680.9	697.3
23	24	1597	-	-	1581	1544.6
24	27	412	-	-	431.1	531.2
25	28	881	-	-	747.7	878.7
26	29	91	82	68	103.8	122.2
27	30	5	-	-	9.5	6.1
28	31	653	-	-	656.6	665.8

As seen in *Table 5*, it is generally observed that the proposed model predicted numbers of defects are closer to the actual number of defects than the predicted number of defects for nine software projects in Chatterjee et al. [46] model. The negative number (-21) is one of the predicted values of the proposed system. This shows that the prediction result of that particular project number is below zero.

Table 6 shows the total number of defects predicted and absolute errors for nine projects at the requirements and design phases undertaken by the proposed work and model of [46].

Table 6. The total number of defects predicted and absolute errors for nine projects.

Project No.	Actual No. of Defects	Predicted No. of Defects by Chatterjee et al. [46]				Predicted No. of Defects by Proposed Model			
		Requirement Phase		Design Phase		Requirement Phase		Design Phase	
		Pv	Absolute Error	Pv	Absolute Error	Pv	Absolute Error	Pv	Absolute Error
1	148	85	63	72	76	145.9	2.1	148.8	0.8
2	31	37	6	12	19	29	2	116.3	85.3
7	204	139	65	168	36	299.1	95.1	249.7	45.7
9	17	41	24	18	1	62.3	45.3	-8.3	25.3
10	29	64	35	43	14	-29	58	72.3	43.3
12	90	219	129	155	65	73.2	16.8	111.3	21.3
15	1768	1946	178	1549	219	1766.3	1.7	1764.5	3.5
17	688	371	317	286	402	762	74	701.7	13.7
29	91	82	9	68	23	103.8	12.8	122.8	31.8
Total			826		855		307.8		270.7

From *Table 6*, PV stands for predicted value.

Table 6 indicates the actual and predicted values together with the absolute error for the nine projects undertaken by the proposed model and model of [46]. This is done to ascertain the accuracy level of each model. From *Table 6*, the total absolute error for [46] is 826 at the requirement and 855 at the design phases while the proposed model provides the least total absolute errors of 307.8 at the requirement and 270.7 at the design phases which show better and accurate predictions. *Figs. 6* and *7* show the plots of the predicted defects values of the study and that of [46] model with the actual defects values for the nine projects considered.

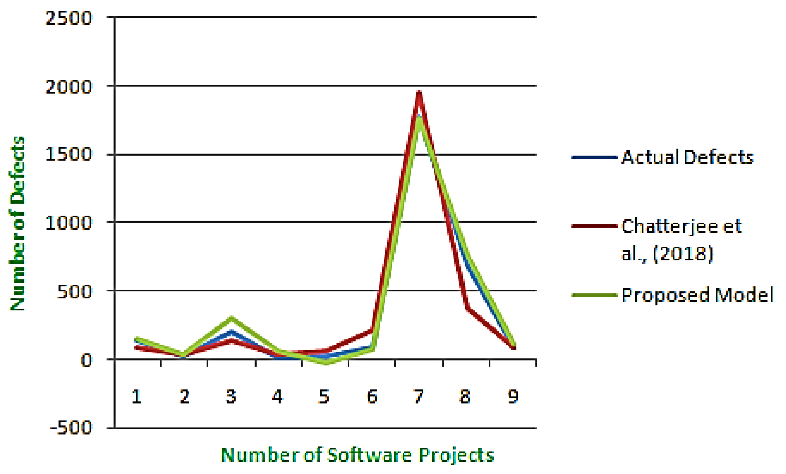


Fig. 6. Plot of total number of defects at the requirements phase for the nine software.

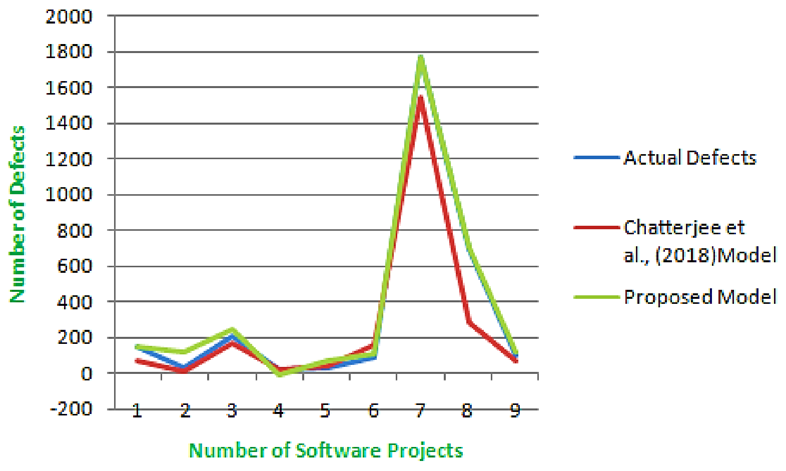


Fig. 7. Plot of total number of defects at the design phase for the nine software projects.

From *Figs. 6* and *7*, it is observed that the proposed model predicted defect values are in close agreement with the actual defect values than [46] predicted defect values, showing a superior prediction. *Table 7* presents the accuracy of the models based on performance measures.

Table 7. Performance comparison of proposed model with Chatterjee et al. [46] model.

Model	RMSE	NRMSE	MMRE	BMMRE	R^2
Chatterjee et al. [46]					
Prediction at requirement phase	132.8230	0.0758	0.6280	0.7247	0.9398
Prediction at design phase	157.0951	0.0897	0.3883	0.6568	0.9158
Proposed model					
Prediction at requirement phase	**71.7201**	**0.0377**	**0.2465**	**0.3680**	**0.9933**
Prediction at design phase	**41.4476**	**0.0218**	**0.1132**	**0.3580**	**0.9983**

Table 7 presents the accuracy of the models based on performance measures. *Table 7* depicts that the proposed model has the smallest errors in terms RMSE, NRMSE, MMRE and BMMRE. *Table 7* also

provides the R^2 values for the models. R^2 for [46] is 0.9398 (93.98%) at the requirement and 0.9158 (91.58%) at the design phases while the proposed model has R^2 values that are closer to 1; 0.9933 (99.33%) at the requirement and 0.9983 (99.83%) at the design phases respectively, showing superior prediction accuracy.

Figs. 8 and *9* present the learning of the optimized IT2FLS showing the relationship between the actual and predicted outputs at the requirements and design phases of the model.

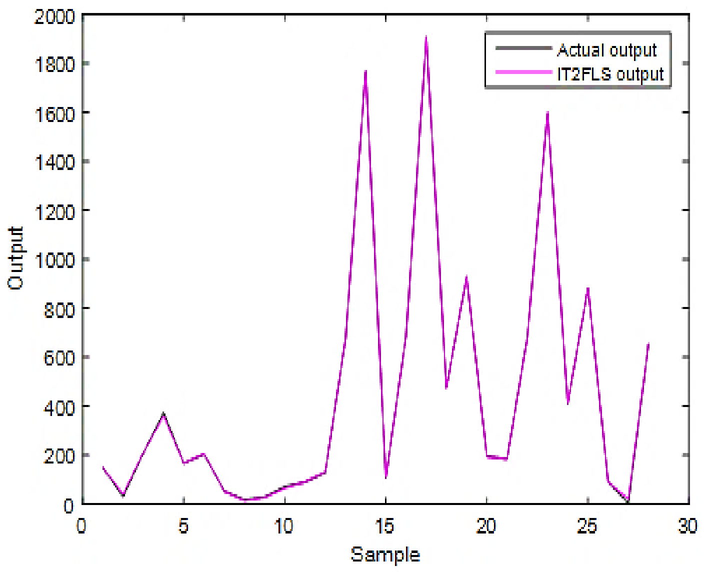


Fig. 8. Actual and predicted outputs of early SRP at the requirements phase.

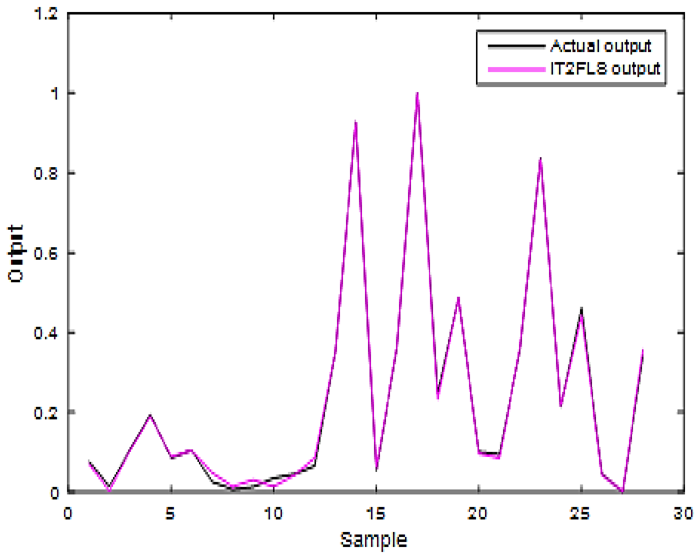


Fig. 9. Actual and predicted outputs of early SRP at the design phase.

Figs. 10 and *11* present the convergence graphs of optimized IT2FLS showing a smooth convergence at the requirement and design phases of the study.

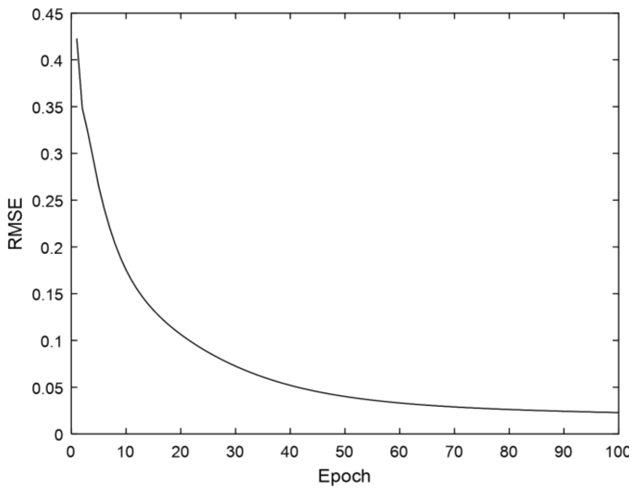


Fig. 10. Convergence graph of optimized IT2FLS-based model at requirements phase.

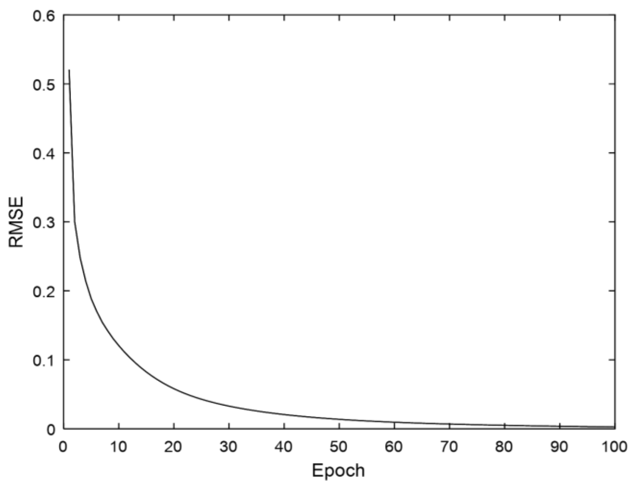


Fig. 11. Convergence graph of optimized IT2FLS-based model at design phase.

Shown in *Figs. 12* and *13* are the graphical views of the different prediction errors with optimized IT2FLS having the least errors.

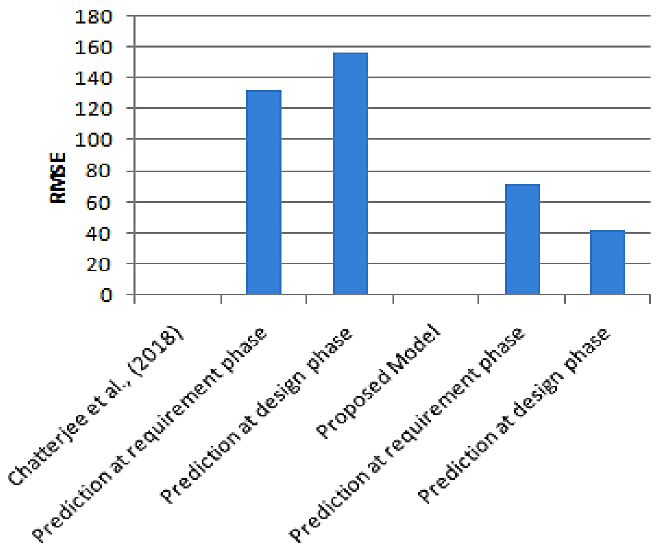
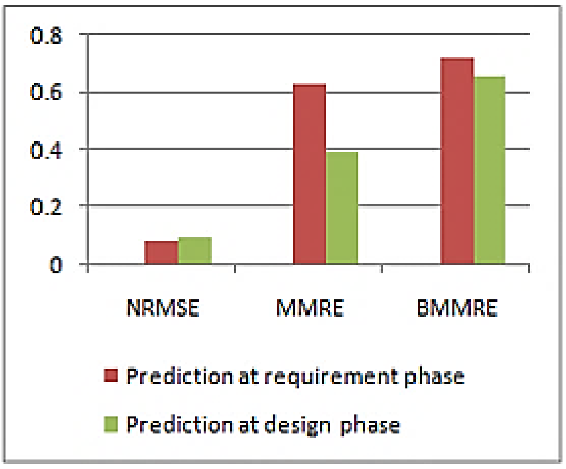
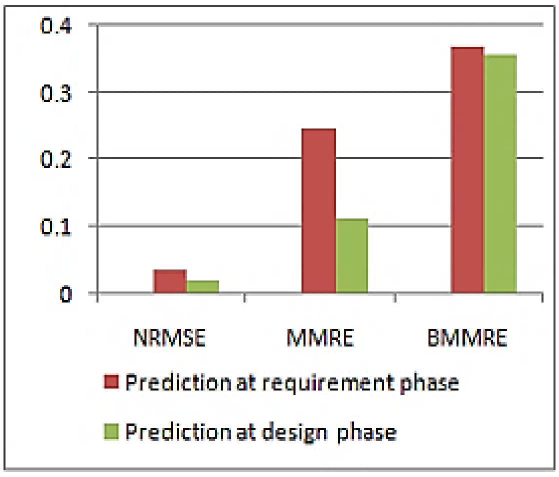


Fig. 12. RMSE of early software defect prediction.



a. Chatterjee et al. [46].



b. Proposed model.

Fig. 13. Early software defect prediction errors (NRMSE, MMRE and BMMRE).

As shown in *Figs. 12* and *13*, the proposed model has the least prediction errors showing a superior performance over the model of [46].

8 | Conclusion

This work develops a model to predict software reliability (number of defects) at the early phase (requirements and design) of software development process. In this work, a-TSK-based interval type 2 fuzzy logic systems, with ANN is employed to handle uncertainty in the prediction of software reliability. The degree of membership grades of the IT2FSs are obtained using interval type 2 gaussian membership function with fixed mean and uncertain standard deviation. The parameters of the IT2FLS membership functions are optimized using GD back-propagation algorithm. Inputs to the system are the top significant reliability relevant software metrics of the early phase and the software size. The model takes into account the uncertainty associated over the assessment of three reliability relevant metrics of requirements phase, namely, RC, RS and RIW; and two reliability relevant metrics of design phase, viz, DTE and PM and software size metric to predict the defect of software. The proposed model made use of real software projects to validate its predictive ability. The performance of the model was evaluated using five performance metrics which include RMSE, NRMSE, MMRE, BMMRE and R^2 and comparison was made with [46] models. The proposed model showed a superior performance over [46] model in terms of prediction accuracy. The model's predicted defects for twenty-eight software projects are found very close to the actual software defect outputs. The proposed model will be useful to the software engineers, designers, stakeholders and researchers with an idea about software reliability in early phase where sufficient data are not available. Precisely, the proposed model will assist software engineers, designers and developers in making informed and useful decisions in the face of uncertainty. In the future, we intend to explore other prediction approaches such as extreme learning machine, extended Kalman filter, support vector machine and hybrid methods in early software defect prediction. We also intend to adopt the generalized interval type-2 fuzzy logic in SRP.

Acknowledgments

The autohors wish to thank Tertiatiary Education Trust Fund (TETFUND) through the establishment of TETFUND centre of excellence in computational intelligence research, university of Uyo and the management of the university of Uyo for providing supports for this research.

Funding

This research is funded by the TETFUND.

Conflicts of Interest

The co-author has seen and agree with the contents of the manuscript and there is no financial interest to report. We certify that the submission is original work and is not under review at any other publication.

References

[1] Lyu, M. R.. (1996). *Handbook of software reliability engineering* (Vol. 222). IEEE computer society press Los Alamitos.

[2] Umoeka, I., Imo Eyoh, E. U., & Akwukwuma, V. (2020). Optimization of Interval Type-2 Fuzzy Logic System for Software Reliability Prediction‖ . *International journal of engineering research and advanced technology*, *6*(11), 1–12.

[3] Kumar, R., Khatter, K., & Kalia, A. (2011). Measuring software reliability: a fuzzy model. *ACM sigsoft software engineering notes*, *36*(6), 1–6.

[4] Board, I. S. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. https://www.informatik.htw-dresden.de/~hauptman/SEI/IEEE_Standard_Glossary_of_Software_Engineering_Terminology .pdf

[5] Arasteh, B. (2018). Software fault-prediction using combination of neural network and Naive Bayes algorithm. *Journal of networking technology*, *9*(3), 94-101.

[6] Kaur, R., & Sharma, E. S. (2018). Various techniques to detect and predict faults in software system: survey. *International journal on future revolution in computer science & communication engineering (IJFRSCE)*, *4*(2), 330–336.

[7] Singh, P., Pal, N. R., Verma, S., & Vyas, O. P. (2016). Fuzzy rule-based approach for software fault prediction. *IEEE transactions on systems, man, and cybernetics: systems*, *47*(5), 826–837.

[8] Eyoh, I. J., Udo, E. N., & Umoeka, I. J. (2021). Software fault prediction based on interval type-2 intuitionistic fuzzy logic system. *international journal of advances in scientific research and engineering*, 7(5). https://www.researchgate.net/publication/352647945

[9] Fan, G., Diao, X., Yu, H., Yang, K., Chen, L., & others. (2019). Software defect prediction via attention-based recurrent neural network. *Scientific programming*, *2019*. https://doi.org/10.1155/2019/6230953

[10] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radliński, Ł., & Krause, P. (2008). On the effectiveness of early life cycle defect prediction with Bayesian nets. *Empirical software engineering*, *13*, 499–537. https://doi.org/10.1007/s10664-008-9072-x

[11] Pandey, A. K., & Goyal, N. K. (2009). A fuzzy model for early software fault prediction using process maturity and software metrics. *International journal of electronics engineering*, *1*(2), 239–245.

[12] Yadav, D. K., Chaturvedi, S. K., & Misra, R. B. (2012). Early Software defects prediction using fuzzy logic. *International journal of performability engineering*, *8*(4). http://www.ijpe-online.com/EN/10.23940/ijpe.12.4.p399.mag

[13] Yadav, H. B., & Yadav, D. K. (2017). Early software reliability analysis using reliability relevant software metrics. *International journal of system assurance engineering and management*, *8*, 2097–2108. https://doi.org/10.1007/s13198-014-0325-3

[14] Rizvi, S. W. A., Khan, R. A., & Singh, V. K. (2016). Software reliability prediction using fuzzy inference system: early stage perspective. *International journal of computer applications*, *145*(10), 16–23.

[15] Rizvi, S. W. A., Singh, V. K., & Khan, R. A. (2017). Early stage software reliability modeling using requirements and object-oriented design metrics: fuzzy logic perspective. *International journal of computer applications*, *162*(2), 44–59.

[16] Zhang, X., & Pham, H. (2000). An analysis of factors affecting software reliability. *Journal of systems and software*, *50*(1), 43–56.

[17] Li, M., Smidts, C., & Brill, R. W. (2000). Ranking software engineering measures related to reliability using expert opinion. *Proceedings 11th international symposium on software reliability engineering* (pp. 246–258). IEEE. https://doi.org/10.1109/ISSRE.2000.885876

[18] Li, M., & Smidts, C. S. (2003). A ranking of software engineering measures based on expert opinion. *IEEE transactions on software engineering*, *29*(9), 811–824.

[19] Kumar, T. R., Rao, T. S., & Hari, CH. V. M. K. (n.d.). A predictive approach to estimate software defects density using probabilistic neural networks for the given software metrics. *Ravi kumar journal of engineering research and application ISSN*, *8*(7), 2248–9622.

[20] Debnath, S. (2021). Fuzzy hypersoft sets and its weightage operator for decision making. *Journal of fuzzy extension and applications*, *2*(2), 163–170.

[21] Khoshgoftaar, T. M., Szabo, R. M., & Guasti, P. J. (1995). Exploring the behaviour of neural network software quality models. *Software engineering journal*, *10*(3), 89–96.

[22] Karunanithi, N., Malaiya, Y. K., & Whitley, L. D. (1991). Prediction of software reliability using neural networks. In *ISSRE* (pp. 124–130). IEEE Computer Society. https://www.researchgate.net/publication/3507480

[23] Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Using neural networks in reliability prediction. *IEEE software*, *9*(4), 53–59.

[24] Tian, L., & Noore, A. (2005). Evolutionary neural network modeling for software cumulative failure time prediction. *Reliability engineering & system safety*, *87*(1), 45–51.

[25] Tian, L., & Noore, A. (2005). On-line prediction of software reliability using an evolutionary connectionist model. *Journal of systems and software*, *77*(2), 173–180.

[26] Oliveira, E., Pozo, A., & Vergilio, S. R. (2006). Using boosting techniques to improve software reliability models based on genetic programming. *2006 18th IEEE international conference on tools with artificial intelligence (ICTAI'06)* (pp. 643–650). IEEE. https://doi.org/10.1109/ICTAI.2006.117

[27] Sharma, T. K., Pant, M., & Abraham, A. (2011). Dichotomous search in abc and its application in parameter estimation of software reliability growth models. *2011 third world congress on nature and biologically inspired computing* (pp. 207–212). IEEE. https://doi.org/10.1109/NaBIC.2011.6089460

[28] Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE transactions on neural networks*, *8*(4), 902–909.

[29] Khoshgoftaar, T. M., Allen, E. B., & Deng, J. (2002). Using regression trees to classify fault-prone software modules. *IEEE transactions on reliability*, *51*(4), 455–462.

[30] Sandhu, P. S., Khullar, S., Singh, S., Bains, S. K., Kaur, M., & Singh, G. (2010). A study on early prediction of fault proneness in software modules using genetic algorithm. *International journal of computer and information engineering*, *4*(12), 1891–1896.

[31] Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., & Thambidurai, P. (2007). Object-oriented software fault prediction using neural networks. *Information and software technology*, *49*(5), 483–492.

[32] Ranjan, P., Kumar, S., & Kumar, U. (2017). Software fault prediction using computational intelligence techniques: A survey. *Indian journal of science and technology*, *10*(18), 1–9.

[33] Erturk, E., & Sezer, E. A. (2016). Iterative software fault prediction with a hybrid approach. *Applied soft computing*, *49*, 1020–1033.

[34] Pandey, A. K., Goyal, N. K., Pandey, A. K., & Goyal, N. K. (2013). Multistage model for residual fault prediction. *Early software reliability prediction: a fuzzy logic approach*, 59–80. https://doi.org/10.1007/978-81-322-1176-1_4

[35] Adak, M. F. (2018). Software defect detection by using data mining based fuzzy logic. *2018 sixth international conference on digital information, networking, and wireless communications (DINWC)* (pp. 65–69). IEEE. https://doi.org/10.1109/DINWC.2018.8356997

[36] Chatterjee, S., & Maji, B. (2016). A new fuzzy rule based algorithm for estimating software faults in early phase of development. *Soft computing*, *20*, 4023–4035.

[37] Kakkar, M., Jain, S., Bansal, A., & Grover, P. S. (2019). Fuzzy logic based model to predict per phase software defect. *International journal of innovative technology and exploring engineering*, *8*(Spec. Issue), 36–41. DOI:10.35940/ijitee.I1006.0789S19

[38] Shankar, K., & Kumar, M. (2020). Software defect prediction using fuzzy logic. *International journal of innovations & advancement in computer science*, *6*(3), 118–124.

[39] Eyoh, I., John, R., & De Maere, G. (2016). Interval type-2 intuitionistic fuzzy logic system for non-linear system prediction. *2016 ieee international conference on systems, man, and cybernetics (SMC)* (pp. 1063–1068). IEEE. https://doi.org/10.1109/SMC.2016.7844383

[40] Mendel, J. M. (2001). On the importance of interval sets in type-2 fuzzy logic systems. *Proceedings joint 9th ifsa world congress and 20th nafips international conference* (Vol. 3, pp. 1647–1652). IEEE.

[41] Hagras, H. A. (2004). A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. *IEEE transactions on fuzzy systems*, *12*(4), 524–539. DOI:10.1109/TFUZZ.2004.832538

[42] Hagras, H. (2007). Type-2 FLCs: A new generation of fuzzy controllers. *IEEE computational intelligence magazine*, *2*(1), 30–43.

[43] Mendel, J. M., & John, R. I. B. (2002). Type-2 fuzzy sets made simple. *IEEE transactions on fuzzy systems*, *10*(2), 117–127.

[44] Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8*(3), 338–353. DOI:10.1016/S0019-9958(65)90241-X

[45] Mendel, J. M., John, R. I., & Liu, F. (2006). Interval type-2 fuzzy logic systems made simple. *IEEE transactions on fuzzy systems*, *14*(6), 808–821.

[46] Chatterjee, S., Maji, B., & Pham, H. (2019). A fuzzy rule-based generation algorithm in interval type-2 fuzzy logic system for fault prediction in the early phase of software development. *Journal of experimental & theoretical artificial intelligence*, *31*(3), 369–391.

[47] Atanassov, K. T., & Atanassov, K. T. (1999). *Intuitionistic fuzzy sets*. Springer.

[48] Moeyersoms, J., de Fortuny, E. J., Dejaeger, K., Baesens, B., & Martens, D. (2015). Comprehensible software fault and effort prediction: A data mining approach. *Journal of systems and software*, *100*, 80–90.

[49] Kayacan, E., & Khanesar, M. A. (2015). *Fuzzy neural networks for real time control applications: concepts, modeling and algorithms for fast learning*. Butterworth-Heinemann.

[50] Yadav, H. B., & Yadav, D. K. (2015). A fuzzy logic based approach for phase-wise software defects prediction using software metrics. *Information and software technology*, *63*, 44–57.

[51] Ibraigheeth, M., & Fadzli, S. A. (2018). Software reliability prediction in various software development stages. *Journal of theoretical and applied information technology*, *96*(7). http://www.jatit.org/volumes/Vol96No7/5Vol96No7.pdf

[52] Kececioglu, O. F., Gani, A., & Sekkeli, M. (2020). Design and hardware implementation based on hybrid structure for MPPT of PV system using an interval type-2 TSK fuzzy logic controller. *Energies*, *13*(7), 1842. https://doi.org/10.3390/en13071842

[53] Tai, K., El Sayed, A. R., Biglarbegian, M., Gonzalez, C. I., Castillo, O., & Mahmud, S. (2016). Review of recent type-2 fuzzy controller applications. *Algorithms*, *9*(2), 1-39.

[54] Begian, M. B., Melek, W. W., & Mendel, J. M. (2008). Parametric design of stable type-2 tsk fuzzy systems. *NAFIPS 2008-2008 annual meeting of the north american fuzzy information processing society* (pp. 1–6). IEEE.

[55] Nie, M., & Tan, W. W. (2008). Towards an efficient type-reduction method for interval type-2 fuzzy logic systems. *2008 IEEE international conference on fuzzy systems (IEEE world congress on computational intelligence)* (pp. 1425–1432). IEEE. https://doi.org/10.1109/FUZZY.2008.4630559

[56] Wu, D., & Tan, W. W. (2005). Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller. *The 14th ieee international conference on fuzzy systems* (pp. 353–358). IEEE.

[57] Menzies, T. (2007). *Promise datasets page*. http://promise.site.uottawa.ca/SERepository/datasets-page.html