Research Article

International Journal of Advanced Technology and Engineering Exploration, Vol 11(113)
ISSN (Print): 2394-5443 ISSN (Online): 2394-7454
http://dx.doi.org/10.19101/IJATEE.2023.10101890

Optimizing software fault prediction using decision tree regression and soft computing techniques

Gurmeet Kaur[1*], Jyoti Pruthi[2] and Parul Gandhi[2]
Research Scholar, Manav Rachna University, Faridabad, Haryana, India[1]
Professor, Manav Rachna International Institute of Research and Studies, Faridabad, Haryana, India[2]

Received: 26-June-2023; Revised: 22-April-2024; Accepted: 26-April-2024

Abstract

This research aims to develop a framework for software fault prediction (SFP) using machine learning techniques. A software fault may be the reason behind the failure of software functioning, and even a minor fault could cause the failure. Efficient SFP improves the overall quality and performance of the software products while streamlining the development process. The framework aims to reduce the cost and time involved in software development while optimizing the reliability of the software. It facilitates quick and efficient testing by identifying the modules that are likely to fail at the early stages of the project. Soft computing techniques provide an easy and effective solution for prediction problems. This study emphasizes the significance of soft computing approaches in SFP and highlights their role in improving computational efficiency, reducing development costs, and enhancing the reliability of software applications. Soft computing-based technique was proposed to address the prediction challenges. A metric suite was suggested, which includes a requirement-based metric and an adoption metric, designed by integrating process metrics of software development phases for fault prediction. It also designs decision tree regression (DTR)-based SFP model that uses these metrics as input and delivers predicted faults as output. The literature review reveals that only a few existing frameworks meet the requirement of implementing SFP models using a broad range of soft computing approaches for the same dataset. The suggested metric suite is validated by computing performance measures such as the area under curve (AUC), F-measure, precision, recall, and accuracy. The high-performance values of the suggested metric suite demonstrate its efficient fault prediction capability. The study also compares the performance of the suggested model with other adaptive neuro fuzzy inference systems (ANFIS), fuzzy-inference systems, and Bayesian-net-based SFP models, measured by root mean square error (RMSE), normalized root mean square error (NRMSE), the mean magnitude of relative error (MMRE), the balanced mean magnitude of relative error (BMMRE), and R-Squared. The suggested model outperforms others, achieving RMSE, MMRE, and R-Squared values of 3.54, 2.04 e-05, and 99.78, respectively. This study presents a highly efficient DTR based SFP model with more fault prediction accuracy than the existing SFP models. Implementation of this model is to significantly reduce costs and improve the time and effort of software development, making it an invaluable tool for software engineers.

Keywords

Software fault prediction, Predicted-fault, Process metrics, Soft-computing, Decision tree regression, Machine learning.

1.Introduction

Soft computing techniques have become indispensable resources for addressing optimization and prediction problems in various computing applications [1]. These approaches provide practical, flexible, and accurate solutions designed specifically to deal with the complexity that arises in real-world situations. These techniques are excellent at handling probabilistic or uncertain data, offering the best answer to a variety of issues in several fields, including biogenetic informatics, telecommunication networks, and computers [1].

Early detection and prediction of software defects have enormous potential to improve the productivity and dependability of the software development process [2]. Through proactive fault detection and resolution across the software development life cycle (SDLC), organizations can save a considerable amount of computation time, cut expenses, and better manage resource allocation [2]. Software defects, which can range from defects to failures, not only prevent programs from functioning as intended but also result in significant time and cost savings if they are not fixed [2].

*Author for correspondence

Efficient software fault prediction (SFP) improves the overall quality and performance of software products while streamlining the development process. Developers can reduce the likelihood of software failures and improve user satisfaction by anticipating and mitigating possible issues before they escalate by applying soft computing-based approaches like machine learning, neural networks, and fuzzy inference systems (FIS).

This study emphasizes the significance of soft computing approaches in SFP and highlights their role in improving computational efficiency, reducing development costs, and enhancing the reliability of software applications. It focuses the significance of proactive fault detection in optimizing the SDLC. It also aims to deliver high-quality software products by examining various soft computing techniques and their applications in fault prediction.

Software system errors are a key concern, and software quality assurance and reliability are crucial to ensure that the program is of excellent quality. Regarding software and its measures, two critical concepts have gained attention from experts. It is a commonly known fact that tasks such as time estimation, practical testing, financial planning, and measurement forecasts ensure quality standards [2]. A software bug is a fault, error, failure, or flaw that occurs during the execution of a software program, which prevents it from giving accurate results [2]. A software fault may be the reason behind the failure of software functioning, and even a minor fault could cause the failure [2]. Software failure refers to the unpredictable results that might arise from external factors and state variables that induce defaults when a utility program is running. If these issues are detected at the early stages of product development, time and worth can be consistently and considerably reduced [3].

It is advisable to use an SFP during software development and apply interaction activities to help anticipate defective modules at the initial phases of software development. Predicting the defective modules in the beginning makes testing easy and quick. Present-day development also raises the requirements for programming standards. Defective software modules cause failures, reduced customer satisfaction, and increased maintenance expenses. The goal of constructing SFP models is to provide sufficient initial assessments of the quality of developing software projects by utilizing courses of action that can be applied right on time for the project life cycle.

Designing SFP models seeks to apply metrics often acquired during the project life cycle to provide sufficient early assessments of software reliability. Additionally, measurements may apply to product development, internal control, programming evaluations, and process creation and verification. Cyclomatic complexity metric (CCM), line of codes (LOC), and Halstead complexity metric (HCM) can be used to analyze the product's complexity. Defect density is a product reliability metric that assesses the defect per kilo LOC or defect per function point. Defect removal efficiency is one of the significant calculations in a programming standard. Class-level metrics were proposed by Chidamber and Kemerer (CK) in 1994 and continue to be used today by a community of experts and software developers [4]. These can be applied in object-oriented programming. These metrics include weighted methods per class (WMC), depth of inheritance tree (DIT), Coupling in between object classes (CBO), response for class (RFC), number of children (NOC), and lack of cohesion of methods (LCOM) [4]. Process metrics are a set of measurements based on data gathered throughout the SDLC. Process metrics include code deltas, churn measures, requirement metrics, and change metrics [4].

SFP intends to predict defect-prone programming modules by using some attributes of the software project. It applies to a known project by preparing an expectation model using project properties or attributes supported by defect data and progressively applying the forecast model to foresee shortcomings for uncertain tasks. The fault prediction model of software has three significant parts, which are.

- SFP methods or features,
- Software fault datasets, and
- Evolution measures of the prediction result

A set of measures based on the data gathered during the first forecast of different programming metrics (such as cyclomatic complexity, LOC, etc.) may be used as an independent parameter. As a result, the required fault predicted (such as the measure of faults, both broken and flawed) functions as a dependent parameter. The software fault information is found in the source code file changed logs and other project files containing information about the execution of the software project. The fault datasets are those where the faulty data is found later in the SDLC. Finally, the performance of the constructed fault forecast model is measured using different execution assessment measurements such as review, exactness, precision, and accuracy.

Soft computing is a collection of computational techniques that provide simple and efficient solutions to prediction-related issues [1]. The primary categories of soft computing techniques include fuzzy computing, neural networks, machine learning, Bayesian nets, evolutionary computing, and others [1].

However, there are still some research challenges in fault prediction. Fault prediction relies heavily on code-related metrics, but there is a need to identify and prioritize other metrics and integrate them to serve as early indicators of potential faults.

It has been noted from the past study that there are very few SFP models developed using customized metrics. Existing models for fault prediction may base on traditional metrics. An optimal combination of process metrics for fault prediction may result in high accuracy in early detection capabilities. The proposed research, therefore, aims to develop a soft computing model incorporating a machine learning approach for SFP using a suggested metric suite.

Nowadays, the prediction problem centers on machine learning methods, and the SFP model affects several phases of software engineering that improve cost, time to completion, and reliability. The objectives of this study include the following steps:

- To study and compare the existing fault prediction techniques.
- To identify, design, and validate a customized metric suite at an early stage of software development.
- To apply the suggested metric suite as input to develop a new SFP model using soft-computing techniques.
- To validate the suggested model using a larger dataset and compare it with the existing SFP models.

The principal advantage of the proposed decision tree regression (DTR)-based SFP model lays in its exceptional accuracy for fault prediction. This accuracy is expected to substantially decrease development costs and enhance efficiency in software development processes.
To achieve the research objectives, specific research questions (RQs) have been formulated that are addressed throughout the paper.
RQ1: How is it proved that the literature study has compared soft computing-based fault prediction techniques for the common component?
RQ2: How is it proved that the suggested metric suite (**M_r and M_a**) efficiently predicts faults?
RQ3: How is the validity of the suggested DTR-based model demonstrated?

The organization of the research paper is as follows: section 2 includes the latest view of the literature on SFP models based on various soft computing techniques and a meta-analysis of the literature. Section 3 describes the design of the suggested metric suite derived by integrating process metrics of phases of the SDLC as inputs and implementing the DTR-based SFP model that accepts the suggested metric suite as inputs (independent variables) and delivers predicted-faults as outputs (a dependent variable). Section 4 describes the prediction and validation outcomes of the suggested metric suite and SFP model and compares them with existing SFPs using soft-computing approaches. Section 5 presents a detailed discussion of the outcomes computed from the suggested metric suite and model for SFP and their validity, as stated in the research objectives. It has been concluded in section 6.

2.Related study

The importance of SFP lies in its ability to identify defective items in software modules before deployment, thereby reducing development costs and time and improving the quality of software packages. Soft computing provides a simple and efficient way to solve prediction problems, making it an ideal approach for SFP. *Table 1* provides a detailed description of the literature survey conducted about SFP, highlighting the various soft-computing methodologies in this area.

Table 1 Relevant literature analysis

Ref. No.	Citation	Objective	Methodology used	Advantages/Limitations
[5]	Liu et al. (2023)	To overcome problems with imbalanced data categorization.	Twin support vector machines (SVM) and clustering	The experimental results demonstrate that the proposed method outperforms existing algorithms in terms of prediction accuracy, robustness, and optimised performance when classifying unbalanced data.

Ref. No.	Citation	Objective	Methodology used	Advantages/Limitations
[6]	Azzeh et al. (2023)	Introduced a novel prediction model to address the imprecision in software measurement that combines fuzzy logic with grey system theory.	Fuzzy logic and gray relational analysis	Furthermore, a sensitivity analysis approach was used to assess the suggested model against various levels of uncertainty. The results demonstrated that the model acts consistently under varying levels of uncertainty.
[7]	Batool and Khan (2023)	Deep learning (DL) techniques were used to forecast software flaws and compare the outcomes with pre-existing models.	CK metrics-based datasets for long short-term memory (LSTM), bidirectional LSTM (BILSTM), and radial basis function network (RBFN)	It has been determined that, in comparison to LSTM and BILSTM, RBFN produces the necessary results more quickly. The suggested model offered a more precise and effective SFP approach and evaluated the model through k-fold cross validation (CV).
[8]	Borandag (2023)	A DL based SFP model was suggested in the extreme programming (XP) work environment to manage the projects efficiently by monitoring the development progress and status of the observed faults.	Incorporated five distinct base classifiers and their RF ensembles from Apache active dataset, extreme programming-test driven development (XP-TDD), eclipse and JIRA, and the classifiers ultimately	Applied a DL neural network, which was made up of convolution neural network (CNN) and the LSTM and BILSTM algorithms. It was determined that using a multi-layered recurrent neural network (RNN) based DL model for fault prediction produced positive results.
[9]	Thirumoorthy (2022)	The fault prediction probability and the feature selection (FS) frequency are used in the suggested work to evaluate the candidate solution's fitness.	Multi-criteria decision making (MCDM) and Rao optimization techniques are the foundations of the hybrid FS (filter–wrapper) approach.	There are three widely used benchmark NASA datasets: PC5, JM1, and KC1. The most significant feature subset for fault prediction with an average accuracy on the benchmark datasets has been used to evaluate the efficacy of the proposed method.
[10]	Goyal (2022)	SFP models applying the FS - evolving populations with mathematical diversification (FS-EPwMD) technique.	Using mathematical diversity for genetic evolution, a unique FS technique is developed.	While input five different classification algorithms—decision tree (DT), K nearest neighbor (KNN), SVM, artificial neural network (ANN) and naive bayes (NB). ANN significantly beat all other SFP models.
[11]	Daoud et al. (2021)	ANNs are among the most popular machine-learning methods for identifying software components that are prone to defects.	NASA information and a cloud-based architecture were utilized to anticipate software defects in real time.	Scaled conjugate gradient, Levenberg-Marquardt, Broyden-Fletcher-Goldfarb-Shanno quasi-Newton, and applied Bayesian regularization (BR). It was discovered that out of all the training functions, the BR performed better.
[12]	Farid et al. (2021)	Suggested a hybrid model that predicts the problematic parts of source code.	CNN with Bi-LSTM applied to PROMISE datasets	The results showed that the hybrid model of convolution neural network (CBIL) model outperforms CNN in terms of average f-measure and RNN in terms of area under curve (AUC), with improvements of 25% and 18%, respectively.
[13]	Zain et al.(2021)	Introduced a unique 1-dimensional convolutional neural network (1D-CNN) architecture that makes use of DL.	Used CNN on NASA datasets along with four baseline classification models DT, SVM, and random forest (RF).	The outcomes showed that an optimal and modest 1D-CNN with a dropout layer beats baseline and state-of-the-art models by 66.79% and 23.88%, respectively, in terms of f-measure.
[14]	Hassouneh et al. (2021)	Suggested novel wrapper algorithms (WOA) variants as to address the problems associated with feature (metric) selection in SFP applications.	Roulette wheel, stochastic universal sampling, random basis, linear position, and process metric.	The author employed 17 publicly available SFP datasets to calculate AUC and running time, and found that the proposed tournament approach outperformed the primary WOA.
[15]	Sharma and	FIS was developed to	Hybrid neuro-fuzzy method	The suggested structure offers enhanced

Ref. No.	Citation	Objective	Methodology used	Advantages/Limitations
	Sangal (2020)	assess how well various measures predict problems in agile software projects.	andProcess metric	precision, mean magnitude of relative error (MMRE), balanced mean magnitude of relative error (BMMRE), root mean square error (RMSE), and normalized root mean square error (NRMSE), particularly for projects involving datasets from the PROMISE repository that have a size of 50 kilo line of code (KLOC) or more.
[16]	Tumar et al. (2020)	Developed a precise framework to foresee programming flaws and supported the FS algorithm in solving classification problems.	Used KNN, linear discriminant analysis (LDA), and DT as three distinct classifiers.	The unbalanced data issue is solved via FS using binary moth fire optimization (BMFO) feature selection, which both improve the data set and compute the average AUC.
[17]	Juneja (2019)	Devised a technique to identify product defects utilising inter-version and inter-project assessment.	Using fuzzy logic as a basis. Metric for products and processes	The recommended method yielded very good results for computing AUC and geometric mean (GM) as well as RMSE for eclipse-java development tools (JDT) and eclipse E plug-in development environment (PDE) based projects.
[18]	Turabieh et al.(2019)	To employ several feature selection (FS) strategies to handle a layered recurrent neural organization (L-RNN) and choose important programming measurements.	The terms binary ant colony optimization (BACO), binary genetic algorithm (BGA), and bound particle swarm optimization (BPSO) are interchangeable and process metrics.	Building a strong classifier using 19 actual projects taken from the PROMISE store and computed AUC as opposed to using a preset feature layout.
[19]	Bilgaiyan et al. (2019)	Performed a near-investigation of the expected for various ANN types.	Highlighted the Elman neural network and the ANN-feed forward back-propagation neural network as two types of neural networks. Products and Processes Measure	For data sets from 21 agile-based projects, the feed forward back-propagation neural network exhibits high algorithm speed, fixed algorithm time, and adaptability to non-critical failures according to Elman organisation.
[20]	Chatterjee and Maji (2018)	To determine a goal evaluation of the programming lifecycle's early vulnerabilities and to project the overall number of errors	Interval type-2 fuzzy reasoning	As a result of non-parametric and non-linear nature of ANN, the suggested model has consolidated ANN, to complete number of defects which help in affirming better expectation and to recognize the association among inputs sources.
[21]	Kalaivani and Beena (2018)	An analysis concentrated on the methods used to introduce the quantity of defects, quantity of alterations, and quantity of newly amended lines.	Object-oriented metrics	The researchers reported that system metrics were frequent and efficient addition to forecast modules that usually encouraged prediction methods.
[22]	Arshad et al. (2018)	Deep Fuzzy C-Mean clustering (DFCM) uses feature compression and semi-supervised DFCM to enhance the quality of the software dataset with imbalanced classes.	Using attribute compression techniques, stochastic under sampling, and semi-supervised DFCM.	An actual software project dataset from eclipse and NASA was used in order to assess the effectiveness of the approach. f-measure and AUC were used to assess the performance of this dataset relative to other more recent classical baseline methods.
[23]	Geng (2018)	Demonstrated a tendency	Process metric, BAPS	The innovation in dimensionality reduction

Ref. No.	Citation	Objective	Methodology used	Advantages/Limitations
		to reduce programming maintenance costs by foreseeing product defect proneness in soft modules.	(bound particle swarm optimisation), and deep neural network (DDN) forecasting	based on BPSO can improve the execution of real-world projects (such as NASA and eclipse) and strengthen organisational structure. It also computes the probability of detection (PD), AUC, and f-measure.
[24]	Singh (2017)	The creation of a model using data from a similar category to forecast an error in a different project falls under this category.	Using rules to help with learning Decision tables, NNge, OneR, Ridor, C4.5, JRip, classification and regression tree (CART), Conjunctive Rule, NB, hybrid classifier decision table naive bayes(DTNB), etc.	In terms of average AUC performance for cases inside projects, C 4.5 has done better. With results of 69%, DTNB outperformed the other rule-based learner when results from multiple projects were compared.
[25]	Dhanajayan and Pillai (2016)	The effective prediction and classification of software faults using Bayesian approaches	Bayesian classification	Using a strong similarity aware clustering algorithm based on the dataset's feature similarity measure and the suggested technique's computed accuracy, precious recall, f-measures, and PD.
[26]	Chatterjee and Maji (2016)	To estimate software flaws using a fuzzy rule-based system in the early stages of development.	Used the linguistic values of software metrics of the requirement analysis Phase	.A fuzzy rule-based strategy was suggested that took into account the size of the programme as well as the weighting of each input software statistic.
[27]	Yadav and Yadav (2015)	Design a multistage fault prediction model	FIS and process metrics	By applying the fuzzy inference tool from Matlab, the author created a model for each software development step. Using a PROMISE dataset, the author obtained results in the form of a defect density value for each stage.
[28]	He et al. (2015)	A research project into software defect prediction using a condensed set of metrics	Process metrics	The study portrayed the process of metric selection was performed through a well-qualified assessment
[29]	Monden et al. (2013)	Examining the effectiveness of methods for dividing up test effort according to fault prediction using a simulation model.	Simulation model	Only when the proper test approach was used and high fault prediction accuracy was a reduction in testing effort realised. A set of modules to be tested, as well as the resource allocation approach and appropriate test strategy.
[30]	Pandey et al. (2013)	Implement a multistage failure prediction model.	FIS and process metrics	At the conclusion of each stage of the SDLC, the output of the model is obtained as a measurement of fault density. The author developed a fuzzy logic model and implemented it in Matlab using data from PROMISE.
[31]	Hall et al. (2012)	Described a quantitative model to examine how measurements are presented in their raw form.	The author employed open source software (OSS) projects and the NASA and PROMISE data sets.	Various studies were conducted utilising OSS projects to examine the capabilities of product measures for programming defect expectation.
[32]	Jin et al. (2012)	Using solely software metrics, the approach finds software modules that are prone to errors.	ANNs and SVMs	They first employed ANNs to eliminate inferior qualities before utilising the SVM to forecast fault-proneness with the features they had chosen. In ANN, the selection of the hidden neuron group is carried out

Ref. No.	Citation	Objective	Methodology used	Advantages/Limitations
				automatically.
[33]	Bishnu and Bhattacherjee (2012)	Design a SFP model using quad tree-based k-means clustering algorithm.	Quad tree-based k-means clustering, logistic regression (LR) technique and NB algorithm	The findings indicate that the quality of machine learning methodology needs to be improved.
[34]	Arisholm et al. (2010)	In order to create forecasting models and discover Java system components with a high defect probability for fault-proneness, compare various data mining and machine learning techniques.	Object oriented metrics has been suggested	Examine numerous approaches for evaluating the models' performance in terms of (i) the suggested cost-effectiveness measure (CE), receiver operating characteristic (ROC) area, accuracy and precision/recall, and confusion matrix.
[35]	Catal and Diri (2009)	To research the effects of FS methods, metrics, and dataset size on the challenge of predicting software faults.	RF algorithms developed using the new artificial immune systems approach to computational intelligence	According to their research, RF outperforms NB as a prediction algorithm for large datasets and is the top option for small datasets.
[36]	Turhan and Bener (2009)	Used software fault data to assess the veracity of the NB assumptions, namely attribute independence and equality in importance.	NB algorithm	It has been demonstrated that the independence assumption of the NB algorithm does not negatively impact primary component analysis (PCA) pre-processing. They employed PD, probability of false alarm (PF), and balancing methods in their investigation.
[37]	Fenton et al. (2008)	Design a multistage fault prediction model	Bayesians Net and process metrics	The values of output variables are computed using a probability distribution, either conditionally or unconditionally, for the node and variables on dataset from PROMISE. Also computed performance measure.
[38]	Khoshgoftaar and Seliya (2007)	Presented a fresh approach to evaluating software quality in the absence of defect information or classifications of programme modules based on their quality.	K-means clustering	The expanding dataset increases the number of clusters and iterations, and this study lacks a clear-cut expert decision-making mechanism. As a result, the expert will need to dedicate a significant amount of time to this technique.
[39]	Koru and Liu (2005)	LOC was used as a predictor, and a set of static measurements were used.	J48 and KStar algorithms	The machine learning methods used by the WEKA are neither inspiring nor depressing. The f-measure was used to assess performance on publicly available NASA datasets, and metrics at the method and class levels were looked at.
[40]	Wang et al. (2004)	To accomplish their aim of improving the understand ability of high-quality prediction models based on neural networks, they employed the clustering genetic algorithm (CGA).	Applied ANNs	When utilising neural networks for prediction, the accuracy of the predictions is higher than when using the CGA rule set.
[41]	Briand et al. (1998)	Coupling and cohesion estimation	Examining the unique statistics, PCA, univariate regression, and related to measure in relation to the defect data.	The author lists a number of recommendations, including the following: method innovation should be given strong emphasis because it has been shown to be important for solid coupling and a steady pointer of defect tracing.

In addition to these studies, there are some other studies [42–44] which are related to the literature discussed.

2.1RQ1. How will it be proved that the literature study compared the various soft computing-based fault prediction techniques for finding a common dataset?

A method used for combining and analyzing data from multiple primary studies is called meta-analysis [45]. Using several data sets, data pre-processing, evaluation systems, and performance statistics makes it challenging to make sense of the many prediction findings [46–49].In addition, no single prediction approach prevails [46].These variations are highlighted even more by the absence of standard reporting procedures [50–55]. Because of this, deciding which fault prediction methodologies to apply and evaluating their chances of success can be challenging for experts and, more importantly, practitioners.

The meta-analysis aims to produce an overall perspective on fault prediction methodologies to address these concerns and provide insight into the most effective approaches. The different parameters that affect the results of the prediction of faults are:

1. The methodology or techniques used to construct and evaluate the prediction model;
2. The features or measures employed as input;
3. The metrics used to evaluate prediction performance;
4. The datasets used to validate prediction performance.

2.2Interpretation of RQ1

Our objective was to select, as thoroughly as possible, significant studies that have performed empirical evaluations of SFP models; additionally, these studies indicate a positive control (i.e., defect- or not-defect-prone). The focus of the meta-analysis is to select the studies from the list of publications that satisfied the criteria of using the same dataset in their proposed work. These studies included the same dataset from the PROMISE repository. These studies served as the basis of the meta-analysis and are mentioned in *Table 2*.

Table 2 Meta-analysis on the basis of Dataset

Ref. No.	Learning technique	Input variable	Performance measures
[15]	Bayesian Net	Process metric	MMRE, BMMRE, MdMRE, R-Squared
[27]	FIS	Process metric	MSE, RMSE, MPE, MAPE
[30]	FIS	Process metric	MMRE, BMMRE
[37]	ANFIS	Process metric	MMRE, BMMRE, RMSE, NRMSE, R-Squared

Where RMSE: root mean square error, NRMSE: normalized root mean square error, MMRE: mean magnitude of relative error, BMMRE: balanced mean magnitude of relative error, MdMRE: median magnitude of relative error, MAPE: mean average precision, MPE: mean percentage error.

The research includes a variety of soft computing-based failure prediction models or frameworks that utilize process or product metrics. The results of the meta-analysis of the literature study indicated a comparison of existing SFP models based on soft computing approaches (Bayesian net, FIS, and ANFIS) for the similar dataset produced by Fenton [37], Panday [30], Yadav [27], and Sharma [15].

Hence, it can be concluded that the literature study compared the soft computing-based fault prediction techniques for a similar dataset as a common component (Dataset). The literature review found that existing SFP models used traditional metrics, and few known models implemented customized metrics. To improve the chances of achieving accurate results in fault prediction, it is essential to incorporate soft computing techniques based on machine learning. An optimal combination of process metrics for fault prediction may result in high accuracy in early detection capabilities.

3.Methods

This section outlines the design of the suggested metric suites and the implementation of the DTR-based SFP model. The methodology for fault prediction is given in *Figure 1*. The model accepted the suggested metric suite as inputs - independent variables - and delivered predicted faults as outputs - a dependent variable.

3.1Suggested metric suite design

Requirement-based metric (M_r) and adoption metric (M_a) were formulated in this section. Process measurements were taken as inputs into the design of Mrand Ma. *Appendix II* shows a subset of a larger real-time dataset from PROMISE repository.

It is a group of variables for every software project, which were further classified into requirements, testing, coding phase metrics, and faults. There are two types of parameters or variables: dependent and independent. For fault prediction, the following variables were considered independent or input: requirements complexity (RC), requirements stability (RS), review, inspection and walkthrough (RIW), design team experience (DTE), process maturity (PM), coding team experience (CTE), defined process followed (DPF), testing team experience (TTE), and stake-holders involvement (SI).

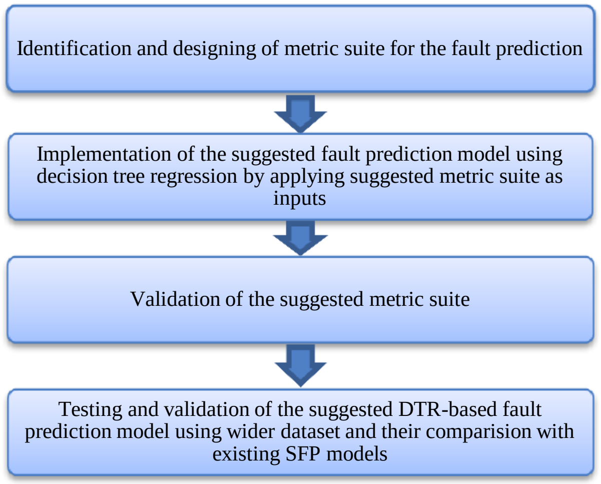
Identification and designing of metric suite for the fault prediction
Implementation of the suggested fault prediction model using decision tree regression by applying suggested metric suite as inputs
Validation of the suggested metric suite
Testing and validation of the suggested DTR-based fault prediction model using wider dataset and their comparision with existing SFP models

Figure 1 Methodology of the suggested framework for fault prediction

The dependent variable was the response or output/target variable which represents the effect of independent variables on the present dataset. Fault was considered a dependent or target variable.

3.1.1Requirement-based metric (M_r)

In the design of M_r, an optimum combination of requirement phase process metrics (such as RC, RS, and RIW) was used, where RC is directly linked to the faults and RIW and RS have an inverse relationship to the faults. Equation 2 provides the requirement-based metric $M_r$for i = 1...N software projects using Equation 1. The M_r inspired requirement-based measure illustrated the relationship between the requirement phase metrics (RC, RS, and RIW).

$$M_r\ (i) = 2 \times RC_r(i) / (RS_r(i) + RIW_r(i)) \qquad (1)$$

$$M_r(i) = \begin{cases} mr(i) & if mr(i) < 0.99 \\ 0.99 & if mr(i) \geq 0.99 \end{cases} \qquad (2)$$

3.1.2Adoption metric (M_a)

For the larger dataset, a different metric based on the process metrics of the SDLC, specifically the design, coding, and testing phases, was proposed. The independent variables in this metric were DTE, PM, CTE, DPF, TTE, and SI; which are inversely related to faults. In this case, all of the independent variables were included in the same location at the same time and were related to the dependent variables in the same way. This metric was termed an "adoption metric," or M_a, and is represented in Equation 4 which has used in Equation 3 for software projects with i =1. N.

$$m_a(i)=(DTE(i)+PM(i)+DPF(i)+CTE(i)+SI(i)+TTE(i))/6 \qquad (3)$$

$$M_a\ (i) = \begin{cases} ma(i) if ma(i) < 0.99 \\ 0.99\ \ if ma(i) \geq 0.99 \end{cases} \qquad (4)$$

Table 3 presents the results of implementing M_r and M_a's Python source code for a subset of 20 software projects from a dataset of PROMISE.

3.2Suggested model

Numerous realistic, precise, and dynamic theories have existed to handle the complexity of real-world problems based on soft computing techniques. The suggested fault prediction model was designed using supervised learning's DTR computation.

3.2.1DTR computing-based SFP model:

To predict faults in software projects, DTR was applied, because the relationship between independent variables and dependent variable is nonlinear. To handle the prediction from nonlinear data, the regression or DT is used. Implicit criteria that apply to both linear and non-linear connections between independent and dependent parameters were used in the design of the DTR algorithm [52]. The suggested model included the pseudo-code for DTR and explained how the algorithm worked to generate regression trees for each project.

1. The entire data used for the training sample was the root node.
2. The value of the independent variables was selected to be exact.
3. Recursively assigning of the documentation was performed using values of independent variables.
4. The DT center node, or root, was determined by addressing the analytical technique. It determined the order in which independent variables were added.

In the development of the SFP model, a regression-based tree was created for each project, with the predicted-faults of each project considered as the target or dependent parameters, while the suggested metric suite of M_r and M_a was applied as an independent parameter. Each regression tree received the values of M_r and M_a as input, and the output was the predicted-faults. *Figure 2* describes the functioning of the DTR-based SFP model.

Table 3 Implementation results of the $M_{r \text{ and }} M_a$

Pr. no.	**RC**	**RS**	**RIW**	**M_r**	**DTE**	**PM**	**CTE**	**DPF**	**TTE**	**SI**	**M_a**
1	0.63	0.63	0.92	0.812903	0.63	0.92	0.86	0.75	0.63	0.86	0.775
2	0.63	0.34	0.75	0.99	0.15	0.75	0.34	0.75	0.34	0.34	0.445
3	0.15	0.34	0.92	0.238095	0.34	0.75	0.86	0.75	0.34	0.86	0.65
4	0.34	0.63	0.75	0.492754	0.63	0.5	0.63	0.5	0.34	0.63	0.538333
5	0.34	0.63	0.75	0.492754	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
6	0.63	0.63	0.75	0.913043	0.63	0.75	0.63	0.5	0.63	0.63	0.628333
7	0.63	0.15	0.75	0.99	0.86	0.75	0.34	0.5	0.63	0.63	0.618333
8	0.63	0.15	0.5	0.99	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
9	0.86	0.63	0.75	0.099	0.63	0.75	0.63	0.75	0.63	0.63	0.67
10	0.63	0.05	0.75	0.99	0.63	0.75	0.63	0.75	0.63	0.86	0.708333
11	0.05	0.34	0.75	0.091743	0.63	0.75	0.63	0.75	0.63	0.86	0.621667
12	0.05	0.34	0.5	0.119048	0.34	0.75	0.34	0.75	0.15	0.34	0.445
13	0.63	0.34	0.75	0.099	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
14	0.86	0.05	0.5	0.099	0.05	0.75	0.05	0.25	0.05	0.63	0.296667
15	0.05	0.34	0.75	0.091743	0.63	0.75	0.63	0.75	0.63	0.63	0.67
16	0.34	0.15	0.5	0.99	0.63	0.75	0.34	0.25	0.34	0.63	0.49
17	0.63	0.34	0.92	0.99	0.15	0.75	0.63	0.75	0.63	0.63	0.59
18	0.34	0.34	0.92	0.539683	0.34	0.75	0.15	0.5	0.34	0.34	0.403333
19	0.34	0.86	0.92	0.382022	0.86	0.75	0.86	0.75	0.86	0.86	0.823333
20	0.05	0.86	0.92	0.05618	0.63	0.75	0.63	0.75	0.63	0.86	0.708333

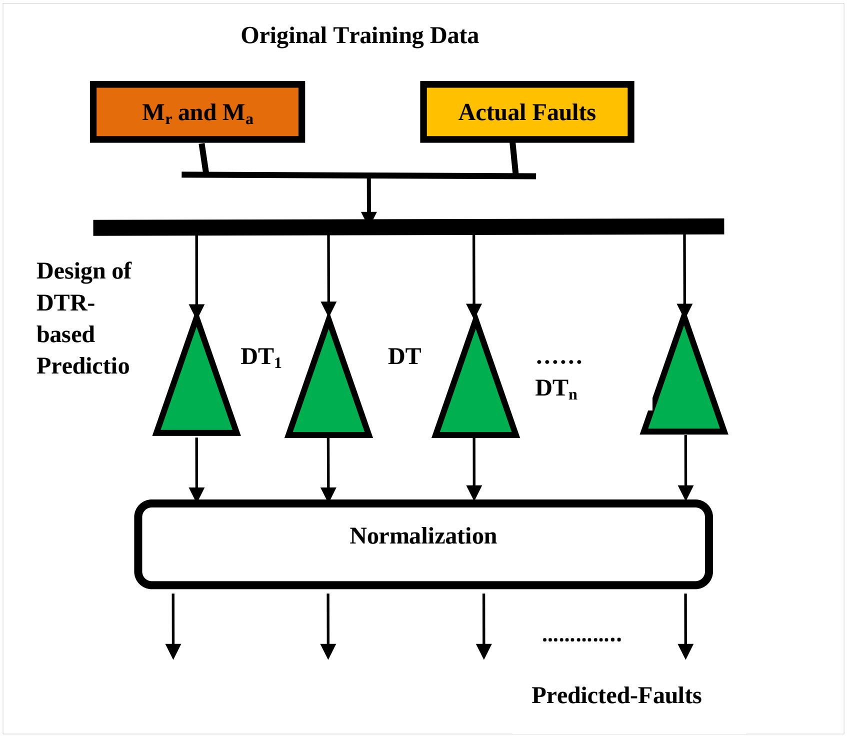

Figure 2 DTR-based SFP model

Algorithm:
For the training dataset, the input states were M_r,M_a, and actual faults, and for the testing dataset, these were M_r and M_a. The predicted-faults were the output state for the training and testing datasets.

The following were the steps in the method for the suggested SFP model that used DTR:

Step 1: The M_r and M_a metric suite of projects was input as an independent parameter or predictor.
Step 2: The project I's actual faults was input as the independent parameter or predictor.
Step 3: The regression tree for projects I.......... 1 to N was generated, where N stands for the total number of projects.
Step 4: To standardize the predicted faults, each tree used Equations 5 and 6 [53].

The predicted-faults for each tree were accepted as standard, represented by dt(I), where I range from 1 to N, and N is the total number of projects.

$$t(I) = [dt(1)+dt(2)+\ldots\ldots.dt(N-1)+dt(N)] \quad (5)$$
$$p(I) = dt(I)/t(I) \quad (6)$$

Figure 3 Illustrates the flowchart of the SFP model that applies DTR.

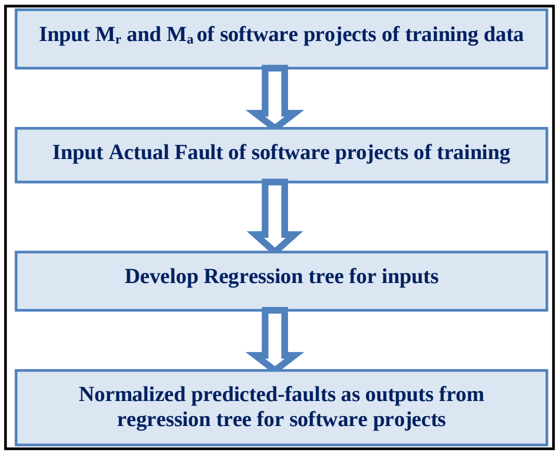
Input M_r and M_a of software projects of training data
Input Actual Fault of software projects of training
Develop Regression tree for inputs
Normalized predicted-faults as outputs from regression tree for software projects

Figure 3 Schematic of the DTR-based SFP model

Data purification and cleaning are necessary to remove inaccurate, imprecise, and incomplete data from datasets and replace missing values. Data integration is the process of merging data from several sources into one dataset, whereas data reduction is a technique for minimizing the amount of data to make the analysis easier. However, the presented dataset was from a single source, and there was no need to apply data reduction. Data transformation is the act of changing data's format or structure. It is achieved through methods of smoothing and normalization. Nevertheless, the suggested algorithm for SFP had no requirement for data transformation. The scaling feature is the final phase of data pre-processing in machine learning. It is a technique for standardizing the independent attributes in the dataset within a specified range. The values of derived attributes (Mr and Ma) were in the range (0 to 0.99) before implementation.

To predict faults, supervised machine learning algorithm (DTR) was utilized, which can used to resolve problems. The model can predict results for new data values that were not used during its training phase. The training and testing datasets for the suggested SFP model had been split into 70% and 30%, respectively, of the dataset. *Figure 4* shows the basic steps involved in machine learning for prediction.

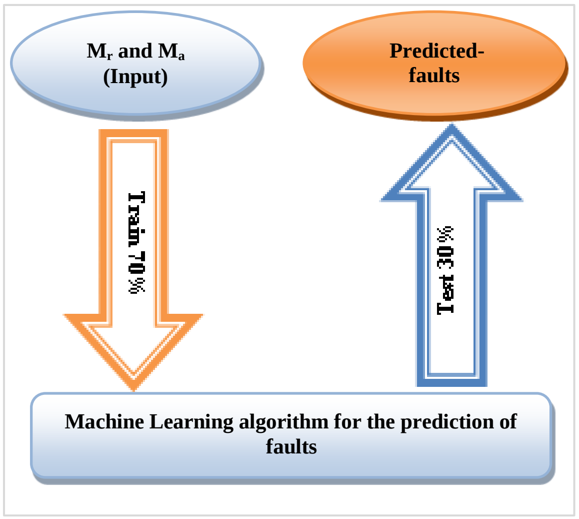

Figure 4 SFP model using machine learning (Original)

Regression is a machine-learning task that involves estimating values. Regression models help us understand the fundamental relationships between inputs and outputs by utilizing the features or properties of input data and their corresponding numeric output values. A DTR consists of three primary nodes:
Root node: It means the goal or decision that must made;
Decision node: It means that sub-node splits and possibilities for each characteristic; and
Leaf node: It provides the outcome subcategories [56].

To identify the position of future splits, the MSE criterion "friedman_mse" was frequently used for node m. Since the desired or dependent parameters were continuous values, the MSE sets the expected value of the final node to the acquired mean value.

Cross validation: One way to assess a machine-learning model's performance is through CV. CV aids in determining an algorithm's optimal performance score, allowing one to choose the most effective solution for the given situation.

Thus, to test the trained model, it needs to use data that the model has never seen before. This dataset is referring as the test dataset. The CV was applied to the dataset using Sklearn of Python, and the results presented the accuracy of the suggested algorithm on a 10-fold CV. The outcomes of the CV showed the accuracy of a larger dataset to be 70.12%.

Hyper parameters are an essential aspect of the DT algorithm, and the DT's hyper-parameters are as follows: max_depth, min_sample_leaf, splitter, max_features, max_leaf_nodes, min_weight_fraction_leaf. As a deeper tree might include more sub-trees to help with decision-making, "max-depth" the maximum tree depth, is a crucial hyper parameter that regulates the complexity of the DT algorithm [57]. The hyperparameters have been adjusted to improve accuracy and prevent overfitting. The hyper-parameters of the suggested DTR-based SFP algorithm were computed as 'max_depth: 3', 'max_features: sqrt', 'max_leaf_nodes: 40', min_samples_leaf: 6', 'min_weight_fraction_leaf: 0.1', and 'splitter: best'.

4. Results

The accuracy obtained after tuning the hyperparameters of the DTR-based SFP algorithm was 99.48%.The accuracy of the suggested DTR-based SFP model for fault prediction is directly related to the size of the software projects, as concluded after evaluating its complexity across different datasets.

Complexity: The accuracy of the suggested DTR-based SFP model for fault prediction is directly related to the size of the software projects, as concluded after evaluating its accuracy across different sets of datasets (up to 60, up to 120, and up to 180 software projects).
Table 4 Size of the software projects correlates with the algorithm's accuracy.

Table 4 Complexity of DTR-based SFP model

Case#	Dataset size	Accuracy
1	Up to 60 software projects	69.84%
2	Up to 120 software projects	94.55%
3	Up to 180 software projects	99.485%

When there were more software projects, the accuracy was higher; when there were fewer software projects, the accuracy was lower.

4.1RQ2:How is it proved that the suggested metric suite (M_r and M_a) efficiently predicts faults?

Regression is the most used statistical method to study the relationship between independent and dependent parameters. In previous research work [44], statistical analysis was conducted to assess the reliability of the metric suite (M_r and M_a). A larger real-time dataset of software projects has been used in an experiment with DTR to calculate the statistical performance measures (precision, recall, F-measure, GM, AUC, and accuracy) of M_r and $M_a$with predicted-faults.

Table 5, summarizes the results of the performance evolution measures of M_r and M_a.

Table 5 Regression analysis results

Performance measures	M_r	M_a
Precision	1	0.5
Recall	1	1
F-measure	1	0.67
Accuracy	0.9513	0.9807
GM	1	0.8185
AUC	98.7185	73.67

Python 3.6 was used to implement the M_r and M_a evaluation. The values of the performance measures of the derived parameter M_r and M_a with faults were in an acceptable range (very high) and indicate that M_r and M_a can be used more equitably for predicting project defects. Consequently, it has been demonstrated that the suggested metric suite (M_r and M_a) can efficiently predict faults.

4.2RQ3 How is the validity of the suggested DTR-based model demonstrated?

To validate the suggested model, the suggested metric suite served as input for the DTR-based SFP model, which generated "predicted-faults" as the output.

4.2.1Prediction analysis

The suggested DTR-based SFP model was tested on a larger, real-time dataset of software projects [51] to validate the prediction results. *Table 6* shows the prediction results for a subset of 20 software projects compared with existing SFP models. *Figure 5* provides a graphical representation of the prediction results computed by the suggested model and their comparison with existing SFP models [15, 27, 30, 37] for a similar dataset.

4.2.2Performance analysis

To confirm the accuracy of the SFP model, the performance metrics RMSE, NRMSE, MMRE, BMMRE, and R-Squared were computed using DTR. [15, 17]. *Table 7* compares the performance measures of the suggested DTR-based SFP model with existing soft computing-based SFP models for a similar dataset. The suggested SFP model has a low value of the RMSE, NRMSE, MMRE, and BMMRE and a high R-Squared value. It indicates that the suggested model is more efficient in predicting faults than existing SFP models [15, 27, 30, 37].

Table 6 DTR-based prediction results and comparisons with existing soft-computing-based SFP models

Pr. No.	**Actual Faults**	**Inputs**		**Predicted-faults**				
		M_r	**M_a**	**Proposed DTR-based model**	**ANFIS-based model [15]**	**FIS-based model [27]**	**FIS-based model [30]**	**Bayesian Net-based model [37]**
1	209	0.812903	0.775	343	191	205	210	254
2	373	0.99	0.445	373	362	----	232	349
3	204	0.238095	0.65	204	185	209	113	262
4	53	0.492754	0.538333	53	52	53	53	48
5	29	0.492754	0.621667	29	36	31	26	203
6	71	0.913043	0.628333	71	80	84	40	51
7	90	0.99	0.618333	254	85	97	176	347
8	129	0.99	0.621667	129	131	142	336	516
9	672	0.099	0.67	672	620	----	697	674
10	1,768	0.99	0.708333	1768	1730	1740	1650	1526
11	109	0.091743	0.621667	109	102	101	127	145
12	688	0.119048	0.445	688	690	733	135	444
13	476	0.099	0.621667	476	422	446	573	581
14	928	0.099	0.296667	928	935	955	869	986
15	196	0.091743	0.67	196	174	192	105	259
16	184	0.99	0.49	184	156	194	291	501
17	680	0.99	0.59	680	686	-----	690	722
18	412	0.539683	0.403333	412	380	----	400	430
19	91	0.382022	0.823333	91	78	91	110	116
20	5	0.05618	0.708333	5	7	5	6	46

Table 7 DTR-based SFP models' performance comparison with soft computing based SFP models

SFPmodels	**RMSE**	**NRMSE**	**MMRE**	**BMMRE**	**R-Squared**
Suggested (DTR-Based)	3.54	0.002008716	2.04E-05	1.98E-05	99.78
ANFIS[15]	48.524	0.0327	0.01646	0.01646	85.4
FIS[27]	55.161	0.0426	0.02171	0.0229	78.9
FIS[30]	350.49	0.627	0.2523	0.6055	56.7
Bayesian Net[37]	324.7	3.454	0.7948	0.7998	50.8719

Table 7 compares the suggested DTR-based SFP model with the present soft computing-based SFP models for RMSE. The suggested DTR-based SFP model had a low RMSE of 3.54 compared to the existing SFP models [15, 27, 30 and 37]. The MMRE computed by DTR-based SFP models and existing models presented in *Table 7*. Compared to the existing SFP models [15, 27, 30, 37], the suggested DTR-based SFP model had a low MMRE of 2.04E-05. The DTR-based SFP model suggested in *Figure 6* had an R-Squared of 99.78%, that is higher than the existing soft computing-based SFP models [15, 27, 30, 37].

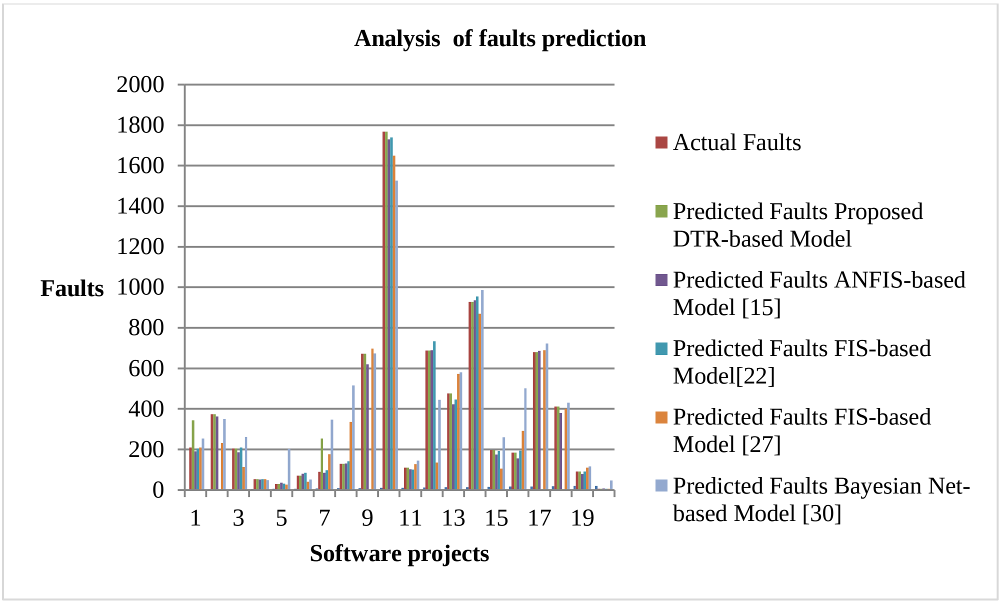

Figure 5 Results of the specified DTR-based SFP model's predictions and a comparison with present SFP models

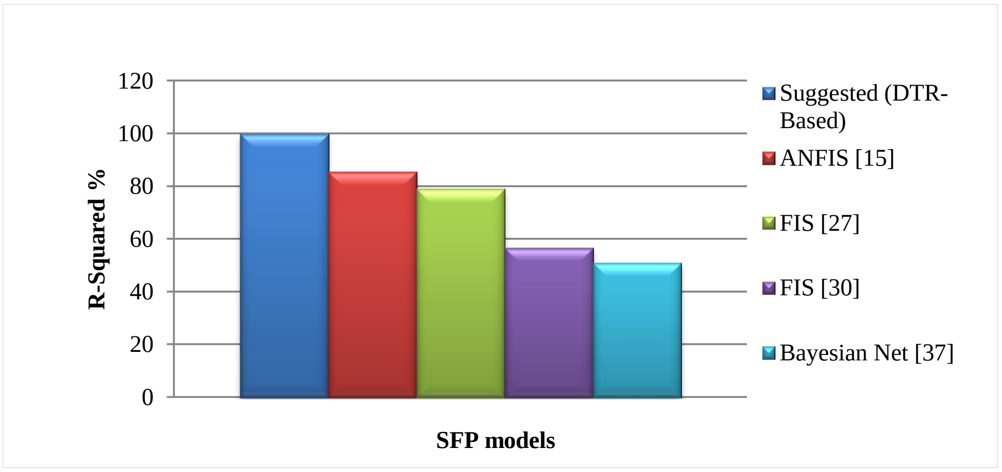

Figure 6 Presenting R-Squared for SFP models

The suggested DTR-based SFP model has been validated for prediction and performance analysis, establishing its validity.

5. Discussion

The study aimed to address specific RQs, with RQ1 derived from the first objective. Through an exploratory review of published works, it found that four studies utilized the same dataset to develop SFP models using different soft computing techniques, including ANFIS [15], FIS [27, 30], and Bayesian classification [37], as outlined in *Table 2*. Based on this evidence, it has been concluded that the literature review compared fault prediction techniques for the same dataset.

The second objective concerns the suggested metric suite (M_r and M_a) and its ability to efficiently predict faults. These metrics were created by combining process metrics for different phases of SDLC and were computed using Equations 2 and 4before being validated using DTR in Python 3.6. The validation process involved measuring AUC, F-measure, precision, recall, GM, and accuracy, and the resulting values are presented in *Table 5*. The performance measures for M_r and M_a with faults were very high, with values of 98.7185, 1.0, 1.0, 1.0, 1.0, and 0.9513 for M_r, and 73.67, 0.67, 0.5, 1.0, 0.8185, and 0.9807 for M_a, respectively. These values fell within an acceptable range, indicating that M_r and M_a were reliable predictors of faults. Thus, we can conclude that the suggested metric suite, M_r and M_a, can efficiently predict faults.

A machine learning-based DTR algorithm is applied to accurately predict faults to achieve the study's third objective. The dataset refined and tested the algorithm using CV, achieving an accuracy of 70.12% at 10-fold. The hyperparameters were fine-tuned to improve accuracy and used the suggested metric suite as inputs to implement the tuned model and deliver "predicted faults" as outputs. *Table 4* illustrates the varying complexity of the suggested DTR-based SFP model, which depends on the size of the software projects. The complexity ranged from 69.84% to 99.485% for 60 to 180 software projects, **demonstrating that the suggested SFP model was effective as the number of software projects increased.**

The third RQ was based on the fourth objective of the study. For generating predictions from the suggested SFP model, the suggested metric suite has been taken as input, and DTR computing has applied to compute the results or predicted faults. The results of the prediction and performance were then compared to existing SFP models that relied on ANFIS [15], FIS [27, 30], and Bayesian classification [37]. These are presented in *Tables 6* and *7*.

Based on the results, it has concluded that

- The suggested DTR-based SFP model performs outstandingly well with a very low RMSE, NRMSE, MMRE, BMMRE, and high R-Squared values of 3.54, 0.002008716, 2.04E-05, 1.98E-05, and 99.78, respectively, compared to the existing ANFIS, FIS, and Bayesian net-based SFP models. This is because the DTR-based SFP model uses derived parameters (M_r and M_a) as independent parameters, while others use the basic parameters of each phase of SDLC and compute fault density at the end of each phase.
- Compared to the FIS, ANFIS, and Bayesian net-based models, the accuracy of the DTR-based SFP model is exceptionally high at 99.48% due to the optimal combination of process metrics and data analysis before implementation, which produces highly accurate results in terms of early detection capabilities.

Therefore, the computed values of predicted faults and performance measures of the DTR-based SFP model confirm its validity. The research paper begins by outlining the objectives of the study, which include designing, developing, and validating a metric suite and SFP model based on DTR for fault prediction. The paper compares the DTR-based SFP model to existing models based on soft computing techniques and concludes that the DTR-based model is highly accurate and valid for fault prediction. The study notes that machine learning-based computing is more efficient for prediction.

5.1 Limitation

The limitation of the study is that the DTR-based SFP model needs to be validated for other datasets. Future work could involve verifying the model on different classifiers such as DL, CNN, and ANN.

A list of abbreviations of terms used in the manuscript has been included in *Appendix I.*

6. Conclusion and future work

The present study aims to develop and validate a machine learning-enabled DTR-based SFP model that accurately predicts faults in software projects. The study involved a comprehensive review of the literature on SFP using soft computing techniques, revealing that a few studies have employed a similar dataset to implement SFP models. The study designed a metric suite that integrated existing process metrics and deployed M_r and $M_a$as independent parameters. $M_r$designed using three process metrics (RC, RS, and RIW), while $M_a$designed using six process metrics (DTE, PM, CTE, DPF, TTE, and SI), with "predicted-faults" as dependent parameters. The metric suite underwent empirical validation through regression analysis and has been implemented using Python with a larger dataset. The performance measures AUC, precision, recall, F-measure, and accuracy for both M_r and M_a with fault were computed, and the results are presented in *Table 5*, confirming the efficacy of the suggested metric suite in predicting faults. The dataset underwent pre-processing as data cleaning, feature scaling, cross-validation, and hyper-parameter tuning before implementation of the suggested model. The DTR-based machine learning algorithm was used to design and implement the model, with M_r and $M_a$as input or independent parameters and predicted-faults as outcomes or dependent parameter. The complexity of the suggested DTR-based SFP model is dependent on the different sizes of software projects, varying from 69.84% to 99.485% for 60 to 180 software projects, as shown in *Table 4*. The interpretation of these results shows that the suggested SFP model works efficiently as software projects increase. The prediction results of the suggested DTR-based SFP model have been compared with existing SFP models for a subset of 20 software projects. The performance measures RMSE, NRMSE, MMRE, BMMRE, and R-Squared have been computed to verify and validate the suggested model using a larger dataset of software projects, and the results are shown in *Table 7*.The suggested DTR-based SFP model performs more accurately, with R-squared, RMSE, and MMRE values of 99.37%, 3.54, and 2.04E-05, respectively, compared to the existing ANFIS, FIS, and Bayesian net-based SFP models. The suggested DTR-based SFP model offers high accuracy for fault prediction, which can significantly reduce the cost, time, and effort required for software development. Our study provides a comprehensive and empirical approach to developing an efficient SFP model using machine learning. The suggested metric suite (M_r and M_a) can predict faults efficiently and be validated empirically. The findings of this study contribute to the development of SFP models and provide a practical and reliable tool for software development.

Future research needs to evaluate the suggested DTR-based SFP system for new datasets using a different classifier, such as CNN, DNN, or KNN.

Acknowledgment

I express my heartfelt gratitude to Dr. Ashwini Kush, Director of the IT Cell at Kurukshetra University, Kurukshetra, Haryana, India, for their kind guidance and help. I am also grateful to Ms. Aastha Bhatia, Assistant Professor of English at Dyal Singh College, Karnal, Haryana, India, who kindly reviewed and edited the language in my manuscript to enhance its quality.

Conflicts of interest

The authors have no conflicts of interest to declare.

Data availability

Due to confidentiality agreements with business associates, the datasets created and processed during the project are not publicly accessible. However, a sample of the data is included in the manuscript. The primary source of the dataset is the PROMISE repository [51].

Author’s contribution statement

All the authors equally contributed in the following stages of this study: study conception and design, data collection, analysis, implementation and interpretation of results, and manuscript preparation.

References

[1] Rathore SS, Kumar S. Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems. 2017; 119:232-56.

[2] Sandhu PS, Khullar S, Singh S, Bains SK, Kaur M, Singh G. A study on early prediction of fault proneness in software modules using genetic algorithm. International Journal of Computer and Information Engineering. 2010; 4(12):1891-6.

[3] Kaur R, Sharma ES. Various techniques to detect and predict faults in software system: survey. International Journal on Future Revolution in Computer Science & Communication Engineering. 2018; 4(2):330-6.

[4] Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 1994; 20(6):476-93.

[5] Liu J, Lei J, Liao Z, He J. Software defect prediction model based on improved twin support vector machines. Soft Computing. 2023; 27(21):16101-10.

[6] Azzeh M, Alqasrawi Y, Elsheikh Y. A soft computing approach for software defect density prediction. Journal of Software: Evolution and Process. 2023; 36(4).

[7] Batool I, Khan TA. Software fault prediction using deep learning techniques. Software Quality Journal. 2023; 31(4):1241-80.

[8] Borandag E. Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques. Applied Sciences. 2023; 13(3):1-21.

[9] Thirumoorthy K. A feature selection model for software defect prediction using binary Rao optimization algorithm. Applied Soft Computing. 2022; 131:109737.

[10] Goyal S. Software fault prediction using evolving populations with mathematical diversification. Soft Computing. 2022; 26(24):13999-4020.

[11] Daoud MS, Aftab S, Ahmad M, Khan MA, Iqbal A, Abbas S, et al. Machine learning empowered software defect prediction system. Intelligent Automation & Soft Computing. 2022; 31(2): 1287:1300.

[12] Farid AB, Fathy EM, Eldin AS, Abd-elmegid LA. Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). Peer J Computer Science. 2021; 7:1-22.

[13] Zain ZM, Sakri S, Asmak INH, Parizi RM. Software defect prediction harnessing on multi 1-dimensional convolutional neural network structure. Computers, Materials & Continua. 2022; 71(1):1521-46.

[14] Hassouneh Y, Turabieh H, Thaher T, Tumar I, Chantar H, Too J. Boosted whale optimization algorithm with natural selection operators for software fault prediction. IEEE Access. 2021; 9:14239-58.

[15] Sharma P, Sangal AL. Building and testing a fuzzy linguistic assessment framework for defect prediction in ASD environment using process-based software metrics. Arabian Journal for Science and Engineering. 2020; 45(12):10327-51.

[16] Tumar I, Hassouneh Y, Turabieh H, Thaher T. Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction. IEEE Access. 2020; 8:8041-55.

[17] Juneja K. A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation. Applied Soft Computing. 2019; 77:696-713.
[18] Turabieh H, Mafarja M, Li X. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. Expert Systems with Applications. 2019; 122:27-42.
[19] Bilgaiyan S, Mishra S, Das M. Effort estimation in agile software development using experimental validation of neural network models. International Journal of Information Technology. 2019; 11(3):569-73.
[20] Chatterjee S, Maji B. A bayesian belief network based model for predicting software faults in early phase of software development process. Applied Intelligence. 2018; 48(8):2214-28.
[21] Kalaivani N, Beena R. Overview of software defect prediction using machine learning algorithms. International Journal of Pure and Applied Mathematics. 2018; 118(20):3863-73.
[22] Arshad A, Riaz S, Jiao L, Murthy A. Semi-supervised deep fuzzy c-mean clustering for software fault prediction. IEEE Access. 2018; 6:25675-85.
[23] Geng W. RETRACTED: cognitive deep neural networks prediction method for software fault tendency module based on bound particle swarm optimization. Cognitive Systems Research. 2018; 5(c):1-12.
[24] Singh P. Comprehensive model for software fault prediction. In international conference on inventive computing and informatics 2017 (pp. 1103-8). IEEE.
[25] Dhanajayan RC, Pillai SA. SLMBC: spiral life cycle model-based bayesian classification technique for efficient software fault prediction and classification. Soft Computing. 2017; 21(2):403-15.
[26] Chatterjee S, Maji B. A new fuzzy rule based algorithm for estimating software faults in early phase of development. Soft Computing. 2016; 20:4023-35.
[27] Yadav HB, Yadav DK. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. Information and Software Technology. 2015; 63:44-57.
[28] He P, Li B, Liu X, Chen J, Ma Y. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology. 2015; 59:170-90.
[29] Monden A, Hayashi T, Shinoda S, Shirai K, Yoshida J, Barker M, et al. Assessing the cost effectiveness of fault prediction in acceptance testing. IEEE Transactions on Software Engineering. 2013; 39(10):1345-57.
[30] Pandey AK, Goyal NK, Pandey AK, Goyal NK. Multistage model for residual fault prediction**.** Early Software Reliability Prediction: a Fuzzy Logic Approach. 2013:59-80.
[31] Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering. 2011; 38(6):1276-304.
[32] Jin C, Jin SW, Ye JM. Artificial neural network-based metric selection for software fault-prone prediction model. IET Software. 2012; 6(6):479-87.
[33] Bishnu PS, Bhattacherjee V. Software fault prediction using quad tree-based K-means clustering algorithm. IEEE Transactions on Knowledge and Data Engineering. 2011; 24(6):1146-50.
[34] Arisholm E, Briand LC, Johannessen EB. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software. 2010; 83(1):2-17.
[35] Catal C, Diri B. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences. 2009; 179(8):1040-58.
[36] Turhan B, Bener A. Analysis of naive bayes assumptions on software fault data: an empirical study. Data & Knowledge Engineering. 2009; 68(2):278-90.
[37] Fenton N, Neil M, Marsh W, Hearty P, Radliński Ł, Krause P. On the effectiveness of early life cycle defect prediction with bayesian nets. Empirical Software Engineering. 2008; 13:499-537.
[38] Khoshgoftaar TM, Seliya N. Software quality classification modeling using the SPRINT decision tree algorithm. International Journal on Artificial Intelligence Tools. 2003; 12(3):207-25.
[39] Koru AG, Liu H. An investigation of the effect of module size on defect prediction using static measures. In proceedings of the 2005 workshop on predictor models in software engineering 2005 (pp. 1-5). ACM.
[40] Wang Q, Yu B, Zhu J. Extract rules from software quality prediction model based on neural network. In 16th international conference on tools with artificial intelligence 2004 (pp. 191-5). IEEE.
[41] Briand LC, Wüst J, Ikonomovski SV, Lounis H. Investigating quality factors in object-oriented designs: an industrial case study. In proceedings of the 21st international conference on software engineering 1999 (pp. 345-54).
[42] Kaur G, Pruthi J. A study of agile-based approaches to improve software quality. International Journal of Computer and Systems Engineering. 2022; 16(5):158-63.
[43] Kaur G, Pruthi J, Gandhi P. Machine learning based software fault prediction models. Karbala International Journal of Modern Science. 2023; 9(2):9.
[44] Kaur G, Pruthi J, Gandhi P. Decision tree regression analysis of proposed metric suite for software fault prediction. SN Computer Science. 2023; 5(1):69.
[45] Keele S. Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report. 2007.
[46] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel

findings. IEEE Transactions on Software Engineering. 2008; 34(4):485-96.
[47] Myrtveit I, Stensrud E, Shepperd M. Reliability and validity in comparative studies of software prediction models. IEEE Transactions on Software Engineering. 2005; 31(5):380-91.
[48] Raeder T, Hoens TR, Chawla NV. Consequences of variability in classifier performance estimates. In international conference on data mining 2010 (pp. 421-30). IEEE.
[49] Song Q, Jia Z, Shepperd M, Ying S, Liu J. A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering. 2010; 37(3):356-70.
[50] Ince DC, Hatton L, Graham-cumming J. The case for open computer programs. Nature. 2012; 482(7386):485-8.
[51] http://promise.site.uottawa.ca/SERepository/datasets-page.html. Accessed 26 March 2024.
[52] Wang S, Yao X. Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability. 2013; 62(2):434-43.
[53] Xu M, Watanachaturaporn P, Varshney PK, Arora MK. Decision tree regression for soft classification of remote sensing data. Remote Sensing of Environment. 2005; 97(3):322-36.
[54] Baştanlar Y, Özuysal M. Introduction to machine learning. miRNomics: MicroRNA Biology and Computational Analysis. 2014; 105-28.
[55] Sarkar D, Bali R, Sharma T. Practical machine learning with python. Book" Practical Machine Learning with Python. 2018; 25-30.
[56] Manias DM, Jammal M, Hawilo H, Shami A, Heidari P, Larabi A, et al. Machine learning for performance-aware virtual network function placement. In global communications conference 2019 (pp. 1-6). IEEE.
[57] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine learning in python. The Journal of Machine Learning Research. 2011; 12:2825-30.

Ms. Gurmeet Kaur is a research scholar in the Department of Computer Science and Technology at Manav Rachna University, Faridabad, Haryana, India. She completed her post-graduation in computer sciences in 2006. She has three research publications in national journals and seven in international journals. She has also successfully published a book chapter. Additionally, she has presented nine research papers at national conferences and three at international conferences.
Email: grmtkaur02@gmail.com

Dr. Jyoti Pruthi is the Joint Director and a Professor in the Department of Computer Science at Manav Rachna University, Faridabad, Haryana, India. She is an experienced academician and researcher with over 18 years of experience spanning research, academics, and industry. Dr. Pruthi has published numerous papers in international journals, conferences, and book chapters. Her research areas include Sentiment Analysis, Data Mining, Software Engineering, Natural Language Processing (NLP), and Prediction Analysis. She is a certified Professional Scrum Master and possesses in-depth working knowledge of Agile frameworks. Dr. Pruthi has played a pivotal role in integrating agility techniques into the education system by establishing the first Agile Classroom in higher education in India. Recently, she was selected from among 200 participants out of 1,500 applicants across India for the AICTE UKIERI Leadership Program conducted by Dudley College, UK. She is also a certified trainer for Infosys and Xebia.
Email: jyoti@mru.edu.in

Dr. Parul Gandhi is a Professor at Manav Rachna International Institute of Research and Studies (MRIIRS), Faridabad, Haryana, India. She holds a Doctorate in Computer Science with a focus on Software Engineering from Guru Jambheshwar University, Hisar. She is also a Gold Medalist in M.Sc. Computer Science. Dr. Gandhi has a strong inclination towards academics and research, boasting over 15 years of experience in academic, research, and administrative roles. She has published more than 30 research papers in reputed international and national journals and conferences. Her research interests include software quality, soft computing, software metrics, component-based software development, data mining, and IoT. Currently, she serves as a Professor at MRIIRS and oversees the university's PhD program. Dr. Gandhi is also an editorial board member and reviewer for various reputed journals and conferences. She has successfully published many book chapters and edited various books in high-indexing databases. Additionally, she is a lifetime member of the Computer Society of India.
Email: parul.sca@mriu.edu.in

Appendix I

S. No.	Abbreviation	Description
1	ANFIS	Adaptive Neuro Fuzzy Inference System
2	ANN	Artificial Neural Network
3	AUC	Area Under Curve
4	BACO	Binary Ant Colony Optimization
5	BILSTM	Bidirectional Long Short-Term Memory
6	BGA	Binary Genetic Algorithm
7	BMFO	Binary Moth Fire Optimization
8	BMMRE	Balanced Mean Magnitude of Relative Error

9	BPSO	Bound Particle Swarm Optimization
10	BR	Bayesian Regularization
11	CART	Classification and Regression Tree
12	CBO	Coupling In Between Object Classes
13	CBIL	Hybrid Model of Convolution Neural Network
14	CCM	Cyclomatic Complexity Metric
15	CE	Cost-Effectiveness Measure
16	CGA	Clustering Genetic Algorithm
17	CK	Chidamber and Kemerer's
18	CNN	Convolution Neural Network
19	CTE	Coding Team Experience
20	1D-CNN	1-Dimensional Convolutional Neural Network
21	CV	Cross Validation
22	DDN	Deep Neural Network
23	DFCM	Deep Fuzzy C-Mean
24	DIT	Depth of Inheritance Tree
25	DPF	Defined Process Followed
26	DL	Deep Learning
27	DTE	Design Team Experience
28	DT	Decision Tree
29	DTNB	Decision Table Naive Bayes
30	DTR	Decision Tree Regression
31	FIS	Fuzzy Inference System
32	FS	Feature Selection
33	FS-EPwMD	Feature Selection - Evolving Populations With Mathematical Diversification
34	GM	Geometric Mean
35	HCM	Halstead Complexity Metric
36	JDT	Java Development Tools
37	KLOC	Kilo Lines of Codes
38	KNN	K Nearest Neighbor
39	LCOM	Lack of Cohesion of Methods
40	LDA	Linear Discriminant Analysis
41	LOC	Line of Codes
42	LR	Logistic Regression
43	L-RNN	Layered Recurrent Neural Organization
44	LSTN	Long Short-Term Memory
45	M_a	Adoption-Based Metric
46	MCDM	Multi-Criteria Decision Making
47	MMRE	Mean Magnitude of Relative Error
48	M_r	Requirement Phase-Based Metric
49	MRE	Magnitude of Relative Error
50	MSE	Mean Square Error
51	NRMSE	Normalized Root Mean Square Error
52	NB	Naive Bayes
53	NOC	Number of Children
54	OSS	Open Source Software
55	PART	Partial C4.5 Rule Based Classifier
56	PCA	Primary Component Analysis
57	PD	Probability of Detection
58	PDE	Plug-in Development Environment
59	PM	Process Maturity
60	PF	Probability of False alarm
61	RBFN	Radial Basis Function Network
62	RC	Requirements Complexity
63	RF	Random Forest
64	RFC	Response for Class
65	RIW	Review, Inspection and Walk-Through
66	RMSE	Root Mean Square Error
67	RNN	Recurrent Neural Network
68	ROC	Receiver Operating Characteristic
69	RQ	Research Question
70	RS	Requirements Stability
71	SDLC	Software Development Life Cycle
72	SFP	Software Fault Prediction
73	SI	Stake-Holders Involvement
74	SVM	Support Vector Machine
75	TTE	Testing Team Experience
76	WMC	Weighted Methods Per Class
77	WOA	Wrapper Algorithms
78	XP	Extreme Programming
79	XP-TDD	Extreme Programming-Test Driven Development

Appendix II

Sr. No.	RS	RC	RIW	DTE	PM	CTE	DPF	TTE	SI	Faults
1	0.86	0.34	0.92	0.86	0.75	0.86	0.75	0.86	0.86	91
2	0.8428	0.3332	0.9016	0.8428	0.735	0.8428	0.735	0.8428	0.8428	89.18
3	0.825944	0.326536	0.883568	0.825944	0.7203	0.825944	0.7203	0.825944	0.825944	87.3964
4	0.809425	0.320005	0.865897	0.809425	0.705894	0.809425	0.705894	0.809425	0.809425	85.64847
5	0.793237	0.313605	0.848579	0.793237	0.691776	0.793237	0.691776	0.793237	0.793237	83.9355
6	0.777372	0.307333	0.831607	0.777372	0.677941	0.777372	0.677941	0.777372	0.777372	82.25679
7	0.761824	0.301186	0.814975	0.761824	0.664382	0.761824	0.664382	0.761824	0.761824	80.61166
8	0.746588	0.295163	0.798675	0.746588	0.651094	0.746588	0.651094	0.746588	0.746588	78.99942
9	0.731656	0.289259	0.782702	0.731656	0.638072	0.731656	0.638072	0.731656	0.731656	77.41944
10	0.717023	0.283474	0.767048	0.717023	0.625311	0.717023	0.625311	0.717023	0.717023	75.87105
11	0.702683	0.277805	0.751707	0.702683	0.612805	0.702683	0.612805	0.702683	0.702683	74.35363
12	0.688629	0.272249	0.736673	0.688629	0.600549	0.688629	0.600549	0.688629	0.688629	72.86655
13	0.674856	0.266804	0.721939	0.674856	0.588538	0.674856	0.588538	0.674856	0.674856	71.40922
14	0.661359	0.261468	0.707501	0.661359	0.576767	0.661359	0.576767	0.661359	0.661359	69.98104
15	0.648132	0.256238	0.693351	0.648132	0.565231	0.648132	0.565231	0.648132	0.648132	68.58142
16	0.635169	0.251113	0.679484	0.635169	0.553927	0.635169	0.553927	0.635169	0.635169	67.20979
17	0.622466	0.246091	0.665894	0.622466	0.542848	0.622466	0.542848	0.622466	0.622466	65.86559
18	0.610017	0.241169	0.652576	0.610017	0.531991	0.610017	0.531991	0.610017	0.610017	64.54828
19	0.597816	0.236346	0.639525	0.597816	0.521351	0.597816	0.521351	0.597816	0.597816	63.25732

Sr. No.	RS	RC	RIW	DTE	PM	CTE	DPF	TTE	SI	Faults
20	0.58586	0.231619	0.626734	0.58586	0.510924	0.58586	0.510924	0.58586	0.58586	61.99217
21	0.34	0.63	0.75	0.15	0.75	0.34	0.75	0.34	0.34	373
22	0.3332	0.6174	0.735	0.147	0.735	0.3332	0.735	0.3332	0.3332	365.54
23	0.326536	0.605052	0.7203	0.14406	0.7203	0.326536	0.7203	0.326536	0.326536	358.2292
24	0.320005	0.592951	0.705894	0.141179	0.705894	0.320005	0.705894	0.320005	0.320005	351.0646
25	0.313605	0.581092	0.691776	0.138355	0.691776	0.313605	0.691776	0.313605	0.313605	344.0433
26	0.307333	0.56947	0.677941	0.135588	0.677941	0.307333	0.677941	0.307333	0.307333	337.1625
27	0.301186	0.558081	0.664382	0.132876	0.664382	0.301186	0.664382	0.301186	0.301186	330.4192
28	0.295163	0.546919	0.651094	0.130219	0.651094	0.295163	0.651094	0.295163	0.295163	323.8108
29	0.289259	0.535981	0.638072	0.127614	0.638072	0.289259	0.638072	0.289259	0.289259	317.3346
30	0.283474	0.525261	0.625311	0.125062	0.625311	0.283474	0.625311	0.283474	0.283474	310.9879
31	0.277805	0.514756	0.612805	0.122561	0.612805	0.277805	0.612805	0.277805	0.277805	304.7682
32	0.272249	0.504461	0.600549	0.12011	0.600549	0.272249	0.600549	0.272249	0.272249	298.6728
33	0.266804	0.494372	0.588538	0.117708	0.588538	0.266804	0.588538	0.266804	0.266804	292.6993
34	0.261468	0.484484	0.576767	0.115353	0.576767	0.261468	0.576767	0.261468	0.261468	286.8454
35	0.256238	0.474794	0.565231	0.113046	0.565231	0.256238	0.565231	0.256238	0.256238	281.1084
36	0.251113	0.465299	0.553927	0.110785	0.553927	0.251113	0.553927	0.251113	0.251113	275.4863
37	0.246091	0.455993	0.542848	0.10857	0.542848	0.246091	0.542848	0.246091	0.246091	269.9765
38	0.241169	0.446873	0.531991	0.106398	0.531991	0.241169	0.531991	0.241169	0.241169	264.577
39	0.236346	0.437935	0.521351	0.10427	0.521351	0.236346	0.521351	0.236346	0.236346	259.2855
40	0.231619	0.429177	0.510924	0.102185	0.510924	0.231619	0.510924	0.231619	0.231619	254.0998
41	0.15	0.63	0.75	0.86	0.75	0.34	0.5	0.63	0.63	90
42	0.34	0.15	0.92	0.34	0.75	0.86	0.75	0.34	0.86	204
43	0.3332	0.147	0.9016	0.3332	0.735	0.8428	0.735	0.3332	0.8428	199.92
44	0.326536	0.14406	0.883568	0.326536	0.7203	0.825944	0.7203	0.326536	0.825944	195.9216
45	0.320005	0.141179	0.865897	0.320005	0.705894	0.809425	0.705894	0.320005	0.809425	192.0032
46	0.313605	0.138355	0.848579	0.313605	0.691776	0.793237	0.691776	0.313605	0.793237	188.1631
47	0.307333	0.135588	0.831607	0.307333	0.677941	0.777372	0.677941	0.307333	0.777372	184.3998
48	0.301186	0.132876	0.814975	0.301186	0.664382	0.761824	0.664382	0.301186	0.761824	180.7118
49	0.295163	0.130219	0.798675	0.295163	0.651094	0.746588	0.651094	0.295163	0.746588	177.0976
50	0.34	0.05	0.75	0.63	0.75	0.63	0.75	0.63	0.63	196