WALAILAK JOURNAL
http://wjst.wu.ac.th Information Technology

Cognitive Estimation of Development Effort, Time, Errors, and the Defects of Software

Amit Kumar JAKHAR* **and Kumar RAJNISH**

Department of Computer Science and Engineering, Birla Institute of Technology, Ranchi, Jharkhand, India

(***Corresponding author's e-mail: amitjakhar69@gmail.com**)

Received: 10 October 2014, Revised: 10 March 2015, Accepted: 10 April 2015

Abstract

In the software industry, measuring the effort and time for developing of software is very challenging. Measuring development effort and time comprises several phases, but measuring the effort in each phase creates problems. It is also observed that estimation of the effort required for developing a project may be over-estimated or under-estimated. It can lead to enormous damage to the organization, with respect to budget and schedule. So, to address the aforementioned, a cognitive technique is proposed for measuring the development effort, time, and errors. After measuring the development effort, machine learning techniques: Bayesian Net, Logistic Regression, Multi-perceptron, SMO, and Lib-SVM are applied for software defect prediction. To estimate the software development effort and defects, NASA PROMISE: CM1, KC3, PC1, PC2, and JM1 datasets and devised datasets (proposed cognitive technique parameters of original datasets) are used. The experimental results of both the experiments prove the goodness of the proposed work of this paper.

Keywords: Cognitive weight, basic control structures, operators, operands, machine learning techniques

Introduction

The software industry grows every day, and estimation of effort for software systems has been investigated by researchers for the last 50 years. However, accurate estimation is still difficult to make. Software requires a lot of man power, money, and time to develop it; what happens when it fails? Due to the frequent failure of software, software engineering became popular [1]. The main objective of software engineering is to develop a quality software product within the pre-determined budget and time frame. So, before beginning to develop a software project, engineers start to estimate the costs of the project with respect to money, time, and the people required to finish it. The majority of researchers pay attention to estimating the required effort to develop software systems. Different effort estimation models have been proposed, and now these models are used in the software industry. Generally, there are 2 types of effort estimation models: Algorithmic models and Non-Algorithmic models. COCOMO, SLIM, Putnam, Function Points (FPs), and Halstead measure are algorithmic models, and Expert Judgment, Analogy, and Machine Learning (like Neural Networks, Fuzzy Logic) models fall in the non-algorithmic category. If the customer requirements are imprecise and dynamic, then the estimated results of these models fall under suspicion.

The most traditional method to measure the required effort for software is Lines of Code (LOC) [2]. This technique is not applicable, because it is programmer dependent. To overcome this limitation, McCabe introduced another measure, called Cyclomatic Complexity (CC) [3]. McCabe measured the complexity of a software system with control flow, but disregards the internal structure of the software. After that, Halstead [4] proposed another method to measure the development effort, time, difficulty, and errors in 1977. Halstead used occurrences of the operators and operands to measure software projects. It

is linked to the internal structure of a software system, like input-output, but disregards the control structure.

Therefore, the complexity of a program depends on both the control flow and the internal structure of the source code, and the cognitive approach can do this job very well. Cognitive Informatics (CI) [5] is used in many research areas for finding a solution of given problems, such as Artificial Intelligence, Software engineering, Cognitive sciences, and many more. It plays a decisive role for measuring the characteristics of software systems. CI [6] found that the complexity of software depends on 3 parameters: Input, Output, and the Architectural flow. It considers both the internal structure and the control flow of the software. The cognitive complexity of software contributed to estimation of the development effort; development cost and time relied on the estimated effort.

In this work, we propose a new cognitive approach to measure software systems in several aspects. The proposed work is based on the number of operands, operators, LOC executables, branch count, and a derived cognitive weight parameter. The whole work is carried out in 2 phases: in the first phase, the above mentioned attributes are used to measure the development effort, time, and the number of errors. In the second phase, software defect prediction is used, which allows us to find out the defected modules before delivery of the software to the customer and before they become severe. Software defect prediction is the most promising area in the field of software engineering [7,8]. The defect prediction models grade the modules as either defected or non-defected; this is very useful, especially for large-scale projects and iterative development environments. It enables the experts to concentrate on the critical portions of the software system which have higher chances of defects. Generally, the defect prediction models are built on the collected historical data of the developed software. The basic hypothesis for software quality prediction is that a software module which is currently under the development stage is likely to have defects if a similar software module with a similar process or product metrics in the previously released projects and which was developed in the same environment was fault prone [9]. Numerous classification techniques are used for software defect prediction; examples include logistic regression [10], optimized set reduction [11], artificial neural networks [12], genetic algorithms [13], and fuzzy classification [14]. A number of defect prediction studies have been made, and all were based on the machine learning approach or the statistical approach. Machine learning algorithms are used in software defect prediction models, for classification and regression. Many research studies employed recent machine learning techniques to enhance the defect prediction rate [15,16]. Preprocessing techniques are also significant in software defect prediction. Before building a defect prediction model, the subsequent techniques may be applied: feature selection technique [17-19], and data normalization and noise reduction [20,21], to improve the performance of machine learning techniques. Several feature selection techniques are used to extract the important features for defect prediction models. With the help of preprocessing techniques, several studies indicated that the prediction performance could be improved [19-21]. In several studies, preprocessing techniques were not applied, because this step is an optional step and can be ignored. However, in this work, the proposed cognitive technique parameters as aforementioned are taken into consideration for defect prediction. Machine learning techniques are applied on these important attributes for defect prediction.

Several challenges were present in the defect prediction models in 2000. The first challenge was that, is it really could be usable for software quality assurance before the software product was delivered to the customer? This can be more helpful when some changes have occurred in the source code. Mockus and Vollta [22] made it possible, and developed a model for changes. This defect prediction model is known as just-in-time (JIT). JIT models have been used by several other researchers in previous years [23-25]. The second problem is related to new projects which are not related with historical projects. Aforesaid, prediction models are based on the historical data of past completed projects; if the historical data is not available, or less available, then it became the one of the most challenging problems in the software industry [26]. To resolve this problem, various cross-project defect prediction models were proposed by the researchers [27,28]. Several researchers are still working to develop a more effective model to find out the defects from the given software. Can these prediction models really help to find the defects and be helpful to the software industry? To find the answer to this question, numerous case studies and real world applications have been accomplished [29,30]. In recent studies, different defect

prediction models have been proposed, with new concepts like the personalized software defect prediction model [31] and the universal model [32]. The performance of any defect prediction model depends on the defect prediction rate and the overall accuracy of certain performance parameters.

The main objective of this work is to measure the development effort, time, number of errors generated in developed code, and prediction of software defects more accurately. In this work, CM1, KC3, PC1, PC2, and JM1 datasets are collected from the NASA PROMISE [33] repository, and every dataset has a distinct number of attributes and defect percentage. After measurement of the result of each dataset, with the help of the proposed cognitive technique, the correlation is measured with the Halstead measure. Because the actual measurement of the projects is not available with the datasets, the Halstead measurement result is used as a gold standard in this study. The correlation shows that a better relation exists between the proposed cognitive technique and the Halstead measure. In addition, 5 machine learning techniques: Bayesian Net, Logistic Regression, Multi-perceptron, SMO, and Lib-SVM, are also applied for software defect prediction. To analyze the performance of machine learning techniques, several performance parameters are used. These parameters are: PD, PF, ACC, TNR, and PRECISION. The WEKA data mining tool is used for prediction of the software defects.

Related works

The related work of this paper is concerned in this section. The Halstead measure, cognitive measures, and machine learning techniques are described here.

Halstead measure

In 1977, Halstead [4] proposed a measure which was built on the number of operators (N1), number of operands (N2), number of unique operators (n1), and number of unique operands (n2). Halstead proposed a number of formulas based on the above mentioned parameters to measure the volume, length, development effort, time, and number of bugs in a given program (**Table 1**).

Table 1 Halstead measure parameters.

n1	Unique operators used in the program
n2	Unique operands used in the program
N1	Total number of operators used in the program
N2	Total Number of operands used in the program

Halstead measure describes the internal structure of the programs. Halstead parameters and their estimated results are given along with the original dataset. Operators and operands are counted from the source code of the software, but how the operators and operands are counted is neither clear nor unambiguous.

Halstead's measures are defined as follows:

Vocabulary: $n = n1 + n2$
Length: $N = N1 + N2$
Volume: $V = N (\log2 (n))$
Difficulty: $D = (n1 / 2) \times (N2 / n2)$
Effort: $E = D \times V$
Dev. Time: $DT = E / 18$
No. of Bugs: $Bugs = V / 3000$

The number of variables, constants, and strings are the part of operands, and the remaining everything else relates to operators.

Illustration of Halstead measure with an example: Assume that the parameters of a given code are as follows:

n1 = 117
n2 = 52
N1 = 784
N2 = 1918

By using these above values the Halstead measures are calculated as follows:

Vocabulary n = 117 + 52 = 169
Length: N = 784 + 1918 = 2702
Volume: V = 2702 (log2 (169)) = 19997.2
Difficulty: D = (117 / 2) × (1918 / 52) = 2157.75
Effort: E = 2157.75 × 19997.2 = 43148907
Dev. Time: DT = 43148907 / 18 =2397161.5 (in seconds)
No. of Bugs: Bugs: = 19997.2 / 3000 = 6.7

Halstead measures are used only after the development of the source code, since they cannot be used in the early stages of the software development life cycle. Generally, Halstead measures are used for measuring the effort required in testing and in the maintenance phase.

Cognitive methods are also developed for estimation of the software systems. Several related techniques are utilized for measuring the complexity of programs. Complexity defines the difficulty level of software that allows us to understand the behavior of the code. Cognitive Functional Size (CFS) [34] was proposed by Wang and Shao to measure the cognitive complexity of software. The functional size of the software depends on the input, output, and architectural control flow. Wang and Shao defined each possible BCSs of the program, and their corresponding cognitive weight is provided in **Table 2**. Cognitive weights indicate the difficulty level in comprehending a particular structure.

Table 2 BCSs with their cognitive weights (W_c).

BCSs Type	BCSs	Weight (W_c)
Sequence	Sequence (seq.)	1
Branch	If then else	2
	Switch - case	3
	For loop	3
Iteration	While loop	3
	Do-while	3
Embedded components	Function call	2
	Recursion	3
Concurrency	Parallel	4
	Interrupt	4

CFS is calculated according to the following Eq. (1).

$$CFS = (N_i + N_o) \times W_c \quad (1)$$

Where N_i: number of inputs, N_o: number of outputs and W_c: BCSs cognitive weight.

Other cognitive techniques have also been developed for measuring the complexity, development time, and understandability of programs. The Cognitive Program Complexity Measure (CPCM) was developed by Misra [35] in 2007 to measure the cognitive complexity of a program. This work shows that

total occurrences of input variables and total occurrences of output variables impose a strong effect on the complexity of software. A New Cognitive Complexity of Program (NCCoP) [36] was proposed by Jakhar and Rajnish in 2014. The number of programs is taken with their source code, and the complexity of programs is measured line by line. The other cognitive method, New Weighted Method Complexity (NWMC) [37] was proposed by Jakhar and Rajnish in 2014. In their work, an experiment was done with the help of 5 postgraduate students of their institute to develop 20 programs. Their development time was observed and the mean of the development time of all the students taken as the actual time required to develop the code, and the number of inputs, number of outputs, and local and global parameters are taken into account to measure the complexity of the program. In addition to measuring NWMC, a constant parameter has been evaluated, so that the actual development time can be achieved with the help of measured cognitive complexity value. The BCSs cognitive weights of **Table 2** were used to measure the cognitive complexity of all the programs.

The proposed work of this paper is completely different from [36,37], because the source code of the projects is not available to analyze the cognitive complexity of the software, and also because we do not have developers who developed the projects. Only some measured values from the developed code are given, along with the original datasets. So, on the basis of these parameter statistics, an attempt is made to develop a cognitive technique which can measure the development effort, time, and number of errors more accurately. Therefore, some significant parameters are extracted from the dataset, such as the number of operands, operators, branch count, LOC executables, and, along with a cognitive weight, these are proposed to measure the software. The Halstead measure is used as a gold standard to compare with the proposed cognitive technique, because other existing cognitive measures require some other attributes, like the number of inputs, number of outputs, etc., which are not given in the original dataset. This is why existing cognitive measures are dropped to further study. The above mentioned parameters are used to predict the software defects as well. Machine learning techniques Bayesian Net, Logistic Regression, Multilayer Perceptron, SMO, and Lib-SVM are applied for defect prediction using original datasets and devised datasets (attributes of the proposed cognitive technique).

Machine learning techniques

Learning is of mainly 2 types: supervised learning and unsupervised learning. Basically, in supervised learning, the supervisor defines the class of each instance and is called learning by example. Unsupervised learning is known as clustering. This technique is the opposite of the supervised learning technique, meaning learning without a supervisor (learning from interpretations). In this paper, Bayesian Net, Logistic Regression, Multilayer Perceptron, SMO, and Lib-SVM are used to predict software defects. In other studies, experiments to analyze the performance of machine learning techniques are done with the WEKA data mining tool [38,39]. Some concerns about machine learning techniques are outlined in the following subsections.

Bayesian Net

Bayesian Net [40] is related to statistically based learning rules. In this technique, the significance of all attributes is equal, and the values of each attribute are calculated separately. Some bays rules are used for classification.

Multilayer perceptron (MLP)

The Multi-Layer Perceptron (MLP) is a feed-forward network. This machine learning technique comes under neural networks. Generally, a neural network model consists of 3 things: an input layer, in which the input is given, hidden (one or more) layers performing the complex mathematical functions, and the output layer that shows the result of the designed architecture [41]. In the neural networks, neurons are used for calculations. Neurons are interconnected through links, and every link has assigned some weight. Each neuron output depends on the activation functions, and the output of the neurons may become the input of the next layer neurons when the network is multi-layer. Activation function may be linear or non-linear, depending on the requirement. Mostly, non-linear activation functions are employed in the hidden layers [42]; these activation functions play a key role in neural networks.

Sequential minimal optimization (SMO)

The SMO technique is used to solve quadratic programming problems. This is utilized to train purpose for support vector machines. The SMO solve optimization problems iteratively; it breaks the large problem into smaller sub-problems, and so on, solving each sub problem analytically.

Logistic regression (LR)

The regression term is defined as measuring and analyzing the relation between independent variables and dependent variables. This can be defined in 2 ways: Linear and Logistic regression. Logistic regression is a generalized form of linear regression [43]. LR is generally used for classification of data which is in the short dimension and has linear boundaries. The main objective of LR is that each variable should be determined correctly. It is also known as a logistic/logit model, which categorizes the target variable in 2 categories, such as light or dark, thin or thick.

Evaluation of proposed work

The whole work of this paper is carried out in 2 phases: in the first phase of study, a cognitive technique is proposed to estimate the development effort, time, and the number of errors. In this study, 4 attributes are used to estimate the development effort, time, and number of errors. These attributes are the number of operands, operators, branch count, and LOC executables. In addition to it, a new parameter, called cognitive weight (W_c) is also proposed, to make the proposed cognitive technique more effective in terms of the control structure of the source code. The proposed technique is the combination of the 4 attributes as discussed above, along with a proposed cognitive weight (W_c), and its flow chart is illustrated in **Figure 1**. Along this, it is also mentioned that the Halstead measure is used as the gold standard to measure the performance of our proposed cognitive technique, as the actual development effort, time, and errors are not mentioned in the original datasets. The development effort, time, and errors are computed using Eqs. (2) - (5) and these equations are described as follows.

Eq. (2) is applied to compute the required effort to develop software. It consists of the number of operands, operators, branch count, LOC executables, and a proposed cognitive weight parameter.

$$Cal\_Effort = ((No.\_of\_operands \times No.\_of\_operators) + (Branch\_count \times W_c) + LOC\_Exe) \quad (2)$$

In Eq. (2), W_c is a proposed cognitive parameter. There are 2 cases to compute the proposed cognitive parameter. In the first case, if the branch count attribute is one, then we assume that, it is an "if" statement in the source code, and the constant "2" is used for the proposed parameter W_c. In the second case, a constant value "3" is used for W_c. This is the mean of all the BCSs cognitive weight, as mentioned in **Table 2**. A generalized formula to compute the cognitive weight is described in Eq. (3).

$$W_c = \begin{cases} 2, & if\ (Branch\_Count < 2) \\ 3, & otherwise \end{cases} \quad (3)$$

With the help of Cal_Effort, the development time is measured, and details are described in Eq. (4). It measures the time in seconds.

$$Cal\_Dev.\_Time = (Cal\_Effort)/10 \quad (4)$$

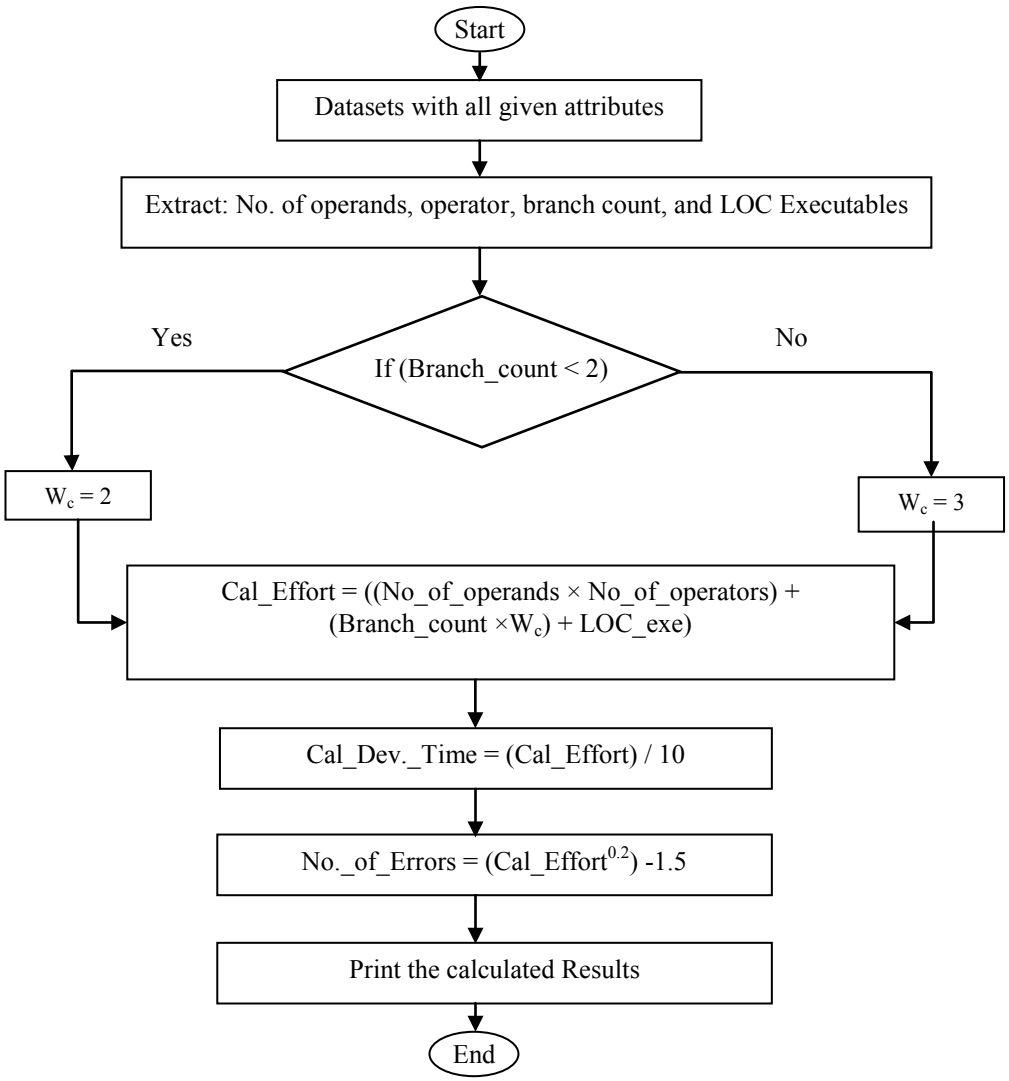

Figure 1 Flow chart of the proposed cognitive technique.

The number of errors that can occur in the developed code is also calculated with the help of Cal_Effort, as given in Eq. (2). Eq. (5) is used to measure the errors.

$$Number\_of\_Errors = (Cal\_Effort)^{0.2} - 1.5 \quad (5)$$

In the second phase of this study, machine learning techniques are applied for defect prediction. These techniques are Bayesian Net, Logistic Regression, Multilayer Perceptron, SMO and Lib-SVM. The proposed cognitive technique of this work consists of 5 parameters to measure the development effort. Therefore, we consider only these most important 5 parameters as attributes, and the aforementioned machine learning techniques are applied for defect prediction. In addition to it, it is also mentioned that all these machine learning techniques are also applied with the original datasets of this study. The result of both the experiments is evaluated on behalf of some performance parameters: PD, PF, ACC, TNR, and PRECISION. The description of these performance parameters is given in the next section. These

performance parameters are evaluated from the outcome of the machine learning techniques. From the experimental results, it is noted that the performance of all machine learning techniques is enhanced with our devised datasets in comparison to the original ones. Alongside this, it is also observed that the LR, MLP, and SMO classifiers provide better results with most of the devised datasets in aspects of several performance parameters.

Performance assessment parameters

To validate the proposed work of this paper, various performance parameters are utilized. Correlation and some other performance parameters are used to analyze the performance of experiments. Correlation is used to validate the existence of proposed cognitive techniques and the performance of machine learning techniques is evaluated using several performance parameters with each dataset (original datasets and devised datasets). Performance parameters used in this study are PD, PF, ACC, TNR, and PRECISION. These parameters are evaluated from a confusion matrix, generated by each machine learning technique. In this work, the modules which are defect prone are treated as "positive" cases and the modules which have no defects are treated as "negative" cases. The confusion matrix categorizes the instances into 4 classes, which are described below and shown in **Table 3**.

- True positive (TP): how many positive cases are predicted correctly as positive.
- True negative (TN): how many negative cases are predicted correctly as negative.
- False positive (FP): the modules which are not defective but predicted as defective.
- False negative (FN): the modules which are defective but predicted as not defective.

Table 3 Confusion matrix for defect prediction.

		Defect prediction?	
Modules with defects?		No	Yes
	No	True negative	False positive
	Yes	False negative	True positive

Performance parameters are analyzed to find out how well the machine learning techniques perform while predicting software defects; some of the parameters are described below.

Probability of Detection (PD) or recall is the percentage of defective modules which is accurately predicted as defective by the machine learning technique. This is calculated as follows:

$$PD = \frac{TP}{TP + FN} \tag{6}$$

Probability of False Alarm (PF) is the percentage of defect free or negative modules that were classified as defect prone modules. It is determined as given in Eq. (7).

$$PF = \frac{FP}{FP + TN} \tag{7}$$

Accuracy (ACC) shows how many modules are predicted correctly by the machine learning technique. TP and TN are correctly classified values. It is measured as follows:

$$ACC = \frac{TP + TN}{TN + TP + FN + FP} \tag{8}$$

The PRECISION indicates the proportion of predicted defect prone modules which are correct. The PRECISION is measured as given in Eq. (9).

$$PRECISION = \frac{TP}{TP + FP} \tag{9}$$

The proportion of the modules that are correctly identified as being defect free is known as True Negative Rate (TNR), calculated as follows:

$$TNR = 1 - PF = \frac{TN}{TN + FP} \tag{10}$$

Experimental results and discussion

This section illustrates the characteristics of the concerned datasets of this work; the experimental results are also discussed in the following subsection.

Data collection

Five NASA PROMISE [33] datasets CM1, PC1, PC2, JM1, and KC3 are used to evaluate the proposed work. All datasets have different number of features, modules, and defect percentage. A description of each dataset is given in **Table 4**.

Table 4 Dataset description.

Projects	Source code	No. of Instances	Percentage defects
CM1	C	327	12.80 %
PC1	C	759	8.10 %
PC2	C	1585	1 %
JM1	C	9371	18.50 %
KC3	C++	200	18 %

The experimental process of this work is done in 2 phases. In the first phase, the development effort, time, and number of errors generated in the code are measured with the help of the proposed cognitive technique. The proposed technique comprises Eqs. (2) - (5), and its flow chart is shown in **Figure 1**. The results of the proposed technique and Halstead measure are correlated with respect to the development effort, time, and number of errors. Correlated results of both techniques are given in **Table 5** along with the datasets. In the second phase of the experiment, software defects are predicted with the help of various machine learning techniques. These techniques are applied to devised datasets, as well as the original datasets. The results of all the machine learning techniques, using both of the datasets, are shown in **Tables 6 - 7**. **Table 6** provides the results of each machine learning technique using original datasets (datasets which are originally reported), whereas **Table 7** contains the results of devised datasets (datasets formed with the help of our proposed cognitive technique). To validate the results of each machine learning technique, the 10 cross-fold method is used.

Discussion

In this research work, a cognitive technique is proposed to estimate the development effort, time, and number of errors. The working of the proposed cognitive technique is illustrated in Eqs. (2) - (5) and in **Figure 1**. In addition, the result of the proposed cognitive technique is correlated with the Halstead measure. Correlation is used because it reveals the relationship existing between the 2 variables up to

some extent. The range of the correlation is always between −1 (indicates no relation exists) to +1 (indicates a strong relation exists). After evaluation of the correlation of the proposed cognitive technique with the Halstead measure, it is found that correlation between the two is in an acceptable level. The correlation of CM1, JM1, and KC3 datasets is near to or greater than 0.95, and the correlation of the remaining PC1 and PC2 datasets is close to 0.90. The correlation results indicate the goodness of the proposed technique. The result of correlation between the 2 techniques of this work is verifiable from **Table 5**.

Table 5 Correlation between Halstead measure and the proposed measure.

Projects	Hal_Effort / Cal_Effort	Hal_Dev_Time / Cal_Dev_Time	Hal_Errors / Cal_Errors
CM1	0.97	0.98	0.92
PC1	0.89	0.89	0.85
PC2	0.87	0.88	0.83
JM1	0.94	0.94	0.76
KC3	0.97	0.97	0.93

In the second phase of the experiment, 5 machine learning techniques are used for software defect prediction. As aforementioned, the parameters of the proposed cognitive technique acted as attributes of our devised dataset, which are further used for defect prediction. Machine learning techniques are applied to the original datasets, as well as the devised datasets. The experimental results of both the datasets (original and devised datasets) are shown in **Tables 6 - 7**, respectively. The results of the machine learning techniques are evaluated on the basis of some performance parameters, which are given as PD, PF, ACC, TNR, and PRECISION. A description of these performance parameters is given in the previous section. The values of these performance parameters are derived from a confusion matrix, which is generated by the each machine learning technique. Each machine learning technique classifies all instances of the datasets as either defective or non-defective. The outcome of the machine learning techniques is structured as shown in **Table 3**. The results of the concerned performance parameters of each machine learning technique, along with the corresponding datasets, are shown in **Tables 6 - 7**. These tables summarize the results of each machine learning technique using original datasets and devised datasets separately. The result of each machine learning technique: Bayesian Net, Logistic Regression, Multilayer Perceptron, SMO, and Lib-SVM is analyzed on the basis of the above mentioned performance parameters. In **Table 7**, calculated results of the performance parameters are indicated in the colored format (*yellow* and *green*). The *yellow color* of **Table 7** indicates that the performance of the machine learning techniques with the original datasets (shown in **Table 6**) and the devised datasets (shown in **Table 7**), is identical, and the *green color* indicates that the devised datasets enhances the performance of the machine learning techniques over the original datasets in perspective to some performance parameters, shown in **Table 7**.

Experimental results for the devised CM1 dataset show that the Accuracy (ACC) and True Negative Rate (TNR) show better performance (shown in **Table 7**) over the original CM1 dataset (shown in **Table 6**). The results of the devised PC1 dataset reveal that the ACC and TNR have been improved, and the PRECISION is further enhanced by applying Logistic Regression and Lib-SVM than with the original PC1 dataset (refer to **Tables 6 - 7**). The result of the devised PC2 dataset shows that the performance of machine learning techniques is enhanced with respect to ACC and TNR parameters, and the Probability of Detection (PD) has also improved by applying Bayesian Net. The PD of the devised JM1 dataset shows the better result of machine learning techniques than the original JM1 dataset, and the ACC and TNR parameters are identical for both the datasets. The performance of the machine learning techniques over the devised KC3 dataset, according to the Probability of False Alarm (PF), ACC, and TNR, shows

they provide better results, and the Multilayer Perceptron model enhances the performance of the PRECISION over the original KC3 dataset. After analyzing the results of each machine learning technique with both the datasets with regard to the performance parameters, it is found that the devised datasets (results shown in **Table 7**) have better capability to predict the software defects than the original datasets (result shown in **Table 6**) when machine learning techniques are applied for the prediction of defects. Please refer to **Tables 5 - 7** for cross verification of the experimental results.

From the above discussion, we can conclude the following:

1. A new cognitive technique is proposed to measure the software development effort, time, and errors with the help of newly generated formulas. The proposed cognitive technique utilizes several parameters, like the number of operands, operators, branch count, LOC executables, and a cognitive weight parameter to measure the software. Their outcome obtains good correlation factor in comparison to the Halstead measure.
2. New datasets have been formed from the original datasets by using parameters of the proposed cognitive technique. Both of the datasets have been used in prediction of the software defects.
3. Five machine learning techniques are applied for defect prediction by using the original datasets, as well as the devised datasets. The performance of each machine learning technique is measured on the basis of some performance parameters like PD, PF, ACC, TNR, and PRECISION.
4. From the analyzed results, it is noted that the devised datasets enhance the performance of machine learning techniques over the original datasets.
5. Among all of the machine learning techniques used in this paper, it is found that LR, MLP, and SMO provide better results in several aspects (refer to **Table 7** for cross verification).

Table 6 Performance parameters of machine learning techniques with the original datasets.

Projects	Performance parameters	Bayesian Net	Logistic Regression	Multilayer Perceptron	SMO	Lib-SVM
CM1	PD	61.90	21.43	4.76	14.29	0.00
	PF	30.88	6.67	5.96	6.32	0.00
	ACC	68.20	84.10	82.56	83.49	87.16
	TNR	69.12	93.33	94.04	93.68	100.00
	PRECISION	22.81	32.14	10.52	25.00	0.00
PC1	PD	75.41	18.03	1.64	14.75	0.00
	PF	29.08	1.86	0.14	2.15	0.87
	ACC	71.28	91.70	91.96	91.17	91.44
	TNR	70.92	98.14	99.86	97.85	99.43
	PRECISION	18.47	45.83	50.00	37.50	0.00
PC2	PD	62.50	0.00	0.00	6.25	0.00
	PF	12.56	0.83	0.00	0.45	0.00
	ACC	87.19	98.17	98.99	98.61	98.99
	TNR	84.44	99.17	100.00	99.55	100.00
	PRECISION	4.83	0.00	0.00	12.50	0.00
JM1	PD	47.63	8.09	5.72	0.17	2.66
	PF	20.86	1.64	1.28	0.00	0.09
	ACC	73.32	81.70	85.54	81.57	81.95
	TNR	79.14	98.36	98.72	100.00	99.91
	PRECISION	34.08	52.83	50.25	100.00	86.79
KC3	PD	36.11	38.89	33.33	5.56	0.00
	PF	14.02	10.98	9.76	0	0.00
	ACC	77.00	80.00	80	83	82.00
	TNR	85.98	89.02	90.24	100	100.00
	PRECISION	36.11	43.75	42.86	100	0.00

Table 7 Performance parameters of machine learning techniques with the devised datasets.

Projects	Performance parameters	Bayesian Net	Logistic Regression	Multilayer Perceptron	SMO	Lib-SVM
CM1	PD	4.76	4.76	0.00	0.00	0.00
	PF	4.91	2.10	0.00	0.00	0.00
	ACC	83.48	85.93	87.16	87.16	87.16
	TNR	95.08	97.89	100.00	100.00	100.00
	PRECISION	12.51	25.00	0.00	0.00	0.00
PC1	PD	63.93	4.92	4.92	0.00	6.56
	PF	39.26	0.43	0.40	0.00	0.86
	ACC	61.00	91.96	91.96	91.97	91.70
	TNR	60.74	99.57	99.87	100.00	99.40
	PRECISION	12.46	50.00	50.00	0.00	40.00
PC2	PD	68.75	0.00	0.00	0.00	0.00
	PF	15.36	0.06	0.00	0.00	0.00
	ACC	84.48	98.93	98.99	98.99	98.99
	TNR	84.64	99.94	100.00	100.00	100.00
	PRECISION	4.87	0.00	0.00	0.00	0.00
JM1	PD	53.01	6.13	7.23	0.29	5.09
	PF	24.62	1.24	1.75	0.00	0.08
	ACC	71.25	81.66	81.46	81.59	81.83
	TNR	75.38	98.76	98.25	100.00	99.85
	PRECISION	32.77	52.74	48.26	100.00	59.06
KC3	PD	13.89	5.56	19.44	5.56	0.00
	PF	6.10	1.83	3.66	0.00	1.22
	ACC	79.50	81.50	83.50	82.00	81.00
	TNR	93.90	98.17	96.34	100.00	98.78
	PRECISION	33.33	40.00	53.80	0.00	0.00

Conclusions and future work

In this paper, CM1, PC1, PC2, JM1, and KC3 datasets from the NASA PROMISE repository are used for measurement of the software development effort, time, and errors. In addition to these, 5 machine learning techniques are used for prediction of the software defects. The whole work is carried out in 2 phases. In the first phase, an attempt is made to develop a new cognitive technique to measure the development effort, time, and number of errors in the software system. The goodness of the proposed technique is measured on the basis of correlation between the proposed cognitive technique and the Halstead measure, and the result is shown in **Table 5**. It indicates that the correlation is in the acceptable level (for some datasets it is near to 0.95, and for remaining, it is near to 0.90). In the second phase, defect prediction has been done with both datasets (original datasets and devised datasets); the result of both of the dataset is shown in **Tables 6** and **7**, respectively. Machine learning technique like LR, MLP, and SMO predict the software defects better with the devised datasets than the original datasets.

The real statistics of the developed projects are not mentioned in the original dataset; only the Halstead metric measured parameters are given. If the actual development effort, time, and number of errors are provided along with the datasets, then this work can be further validated.

References

[1] RS Pressman. *Software Engineering-A Practitioner's Approach*. 5th ed. McGraw Hill, New York, 2002.
[2] JP Kearney, RL Sedlmeyer, WB Thompson, MA Gary and MA Adler. Software complexity measurement. *Comm. ACM* 1986; **28**, 1044-50.
[3] TJ McCabe. A complexity measure. *IEEE Trans. Software Eng.* 1976; **2**, 308-20.
[4] MH Halstead. *Elements of Software Science*. Elsevier, New York, 1977.
[5] Y Wang. On cognitive informatics. *In*: Proceedings of the 1st IEEE International Conference on Cognitive Informatics. Calgary, Alta, Canada, 2002, p. 34-42.
[6] Y Wang. On the cognitive informatics foundations of software engineering. *In*: Proceedings of the 3rd IEEE International Conference on Cognitive Informatics. Victoria, BC, Canada, 2004, p. 22-32.
[7] MD Ambros, M Lanza and R Robbes. An extensive comparison of bug prediction approaches. *In*: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories. Cape Town, SA, 2010, p. 31-41.
[8] T Lee, J Nam, DG Han, S Kim and HP In. Micro interaction metrics for defect prediction. *In*: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA, 2011, p. 311-21.
[9] TM Khoshgoftaar, FD Ross, R Munikoti, N Goel, EB Allen and A Nandi. Predicting fault-prone modules with case-based reasoning. *In*: Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering. Albuquerque, NM, USA, 1997, p. 27-35.
[10] VR Basili, LC Briand and WL Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Software Eng.* 1996; **22**, 751-61.
[11] LC Briand, VR Basili and CJ Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Trans. Software Eng.* 1993; **19**, 1028-44.
[12] TM Khoshgoftaar and DL Lanning. A neural network approach for early detection of program modules having high risk in the maintenance phase. *J. Syst. Software* 1995; **29**, 85-91.
[13] D Azar, S Bouktif, B Kegl, H Sahraoui and D Precup. Combining and adapting software quality predictive models by genetic algorithms. *In*: Proceedings of the 17th IEEE International Conference on Automated Software Engineering. Edinburgh, UK, 2002, p. 285-8.
[14] C Ebert. Classification techniques for metric-based software development. *Software Qual. J.* 1996; **5**, 255-72.
[15] M Li, H Zhang, R Wu and ZH Zhou. Sample-based software defect prediction with active and semi-supervised learning. *Automat. Software Eng.* 2012; **19**, 201-30.
[16] B Turhan, AT Msrl and A Bener. Empirical evaluation of the effects of mixed project data on learning defect predictors. *Inform. Software Tech.* 2013; **55**, 1101-18.
[17] J Nam, SJ Pan and S Kim. Transfer defect learning. *In*: Proceedings of the 13th ACM International Conference on Software Engineering. Piscataway, NJ, USA, 2013, p. 382-91.
[18] S Kim, H Zhang, R Wu and L Gong. Dealing with noise in defect prediction. *In*: Proceedings of the 33rd IEEE International Conference on Software Engineering. New York, USA, 2011, p. 481-90.
[19] S Shivaji, E Whitehead, R Akella and S Kim. Reducing features to improve code change-based bug prediction. *IEEE Trans. Software Eng.* 2013; **39**, 552-69.
[20] T Menzies, J Greenwald and A Frank. Data mining static code attributes to learn defect predictors. *IEEE Trans. Software Eng.* 2007; **33**, 2-13.
[21] R Wu, H Zhang, S Kim and SC Cheung. Recovering links between bugs and changes. *In*: Proceedings of the 19th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA, 2011, p. 15-25.
[22] A Mockus and LG Votta. Identifying reasons for software changes using historic databases. *In*: Proceedings of the IEEE International Conference on Software Maintenance. San Jose, CA, USA, 2000, p. 120-30.

[23] T Fukushima, Y Kamei, S McIntosh, K Yamashita and N Ubayashi. An empirical study of just-in-time defect prediction using cross-project models. *In*: Proceedings of the 11th ACM Working Conference on Mining Software Repositories. New York, USA, 2014, p. 172-81.
[24] Y Kamei, E Shihab, B Adams, AE Hassan, A Mockus, A Sinha, and N Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. Software Eng.* 2013; **39**, 757-73.
[25] S Kim, EJ Whitehead and Y Zhang. Classifying software changes: Clean or buggy? *IEEE Trans. Software Eng.* 2008; **34**, 181-96.
[26] T Zimmermann, N Nagappan, H Gall, E Giger and B Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. *In*: Proceedings of the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA, 2009, p. 91-100.
[27] Y Ma, G Luo, X Zeng and A Chen. Transfer learning for cross-company software defect prediction. *Inform. Software. Tech.* 2012; **54**, 248-56.
[28] B Turhan, T Menzies, AB Bener and J Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empir. Software Eng.* 2009; **14**, 540-78.
[29] C Lewis, Z Lin, C Sadowski, X Zhu, R Ou and EJ Whitehead. Does bug prediction support human developers? Findings from a Google case study. *In*: Proceedings of the 35th IEEE International Conference on Software Engineering. San Francisco, CA, USA, 2013, p. 372-81.
[30] F Rahman, S Khatri, ET Barr and P Devanbu. Comparing static bug finders and statistical prediction. *In*: Proceedings of the 36th ACM International Conference on Software Engineering. New York, USA, 2014, p. 424-34.
[31] T Jiang, L Tan and S Kim. Personalized defect prediction. *In*: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. Silicon Valley, CA, USA, 2013, p. 279-89.
[32] F Zhang, A Mockus, I Keivanloo and Y Zou. Towards building a universal defect prediction model. *In*: Proceedings of the 11th ACM Working Conference on Mining Software Repositories. New York, USA, 2014, p. 182-91.
[33] PROMISE Software Engineering Repository, Available at: http://promise.site.uottawa.ca/ SERepository, accessed June 2014.
[34] Y Wang and J Shao. Measurement of the cognitive functional complexity of software. *In*: Proceedings of the 2nd IEEE International Conference on Cognitive Informatics. London, UK, 2003, p. 67-74.
[35] S Misra. Cognitive program complexity measure. *In*: Proceedings of the 6th IEEE International Conference on Cognitive Informatics. Lake Tahoo, CA, USA, 2007, p. 120-5.
[36] AK Jakhar and K Rajnish. A new cognitive approach to measure the complexity of software's. *Int. J. Software Eng. Its Appl.* 2014; **8**, 185-98.
[37] AK Jakhar and K Rajnish. Measuring complexity, development time and understandability of a program: A cognitive approach. *Int. J. Inform. Tech. Comput. Sci.* 2014; **6**, 53-60.
[38] K Yugal and G Sahoo. Analysis of parametric & non parametric classifiers for classification technique using WEKA. *Int. J. Inform. Tech. Comput. Sci.* 2012; **4**, 43-9.
[39] K Yugal and G Sahoo. Study of parametric performance evaluation of machine learning and statistical classifiers. *Int. J. Inform. Tech. Comput. Sci.* 2013; **5**, 57-64.
[40] N Fenton, M Neil, W Marsh, P Hearty, L Radliski and P Krause. On the effectiveness of early life cycle defect prediction with Bayesian nets. *Empir. Software Eng*. 2008; **13**, 499-537.
[41] MJ Diamantopoulou, VZ Antonopoulos and DM Papamichai. The use of a neural network technique for the prediction of water quality parameters. *Oper. Res.* 2005; **5**, 115-25.
[42] WS Sarle. Neural networks and statistical models. *In*: Proceedings of the 19th Annual SAS Users Group International Conference. Cary, NC, USA, 1994, p. 1538-50.
[43] RL de Mantaras and E Armengol. Machine learning from example: Inductive and Lazy methods. *Data Knowl. Eng.* 1998; **25**, 99-123.