IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 4, 2017, accepted April 27, 2017, date of publication June 5, 2017, date of current version July 17, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2707539

A Comparison Among ARIMA, BP-NN, and MOGA-NN for Software Clone Evolution Prediction

JAYADEEP PATI, BABLOO KUMAR, DEVESH MANJHI, AND K K SHUKLA
Department of Computer Science and Engineering, IIT Varanasi, Varanasi 221005, India
Corresponding author: Jayadeep Pati (jayadeep.rs.cse12@iitbhu.ac.in)

ABSTRACT Software evolution continues throughout the life cycle of the software. During the evolution of software system, it has been observed that the developers have a tendency to copy the modules completely or partially and modify them. This practice gives rise to identical or very similar code fragments called software clones. This paper examines the evolution of clone components by using advanced time series analysis. In the first phase, software clone components are extracted from the source repository of the software application by using the abstract syntax tree approach. Then, the evolution of software clone components is analyzed. In this paper, three models, Autoregressive Integrated Moving Average, back propagation neural network, and multi-objective genetic algorithm-based neural network, have been compared for the prediction of the evolution of software clone components. Evaluation is performed on the large open-source software application, ArgoUML. The ability to predict the clones helps the software developer to reduce the effort during software maintenance activities.

INDEX TERMS Software clones, software clone evolution, ARIMA, back propagation, MOGA-NN, time series analysis.

I. INTRODUCTION

The primary responsibility of a software manager is not only to ensure software quality and standard service levels but also to minimize the cost associated with maintenance effort [1]. The necessary resources must be available to resolve the defects and promptly make the system operate as expected to achieve the customer satisfaction. It is essential to have a predictive model which can help in reducing the maintenance effort with the proper resources (maintenance staff and critical resources) utilization and planning. The prior information regarding the software clone evolution will be valuable information which will reduce the maintenance effort.

Modern Software is agile [2] i.e. they require continuous changes to adopt the changing requirements. This rapid change of requirements leads to software code fragments being slightly modified or copied to have higher functionalities. This slightly modified or similar code fragments are essential in reducing the effort required in maintenance if we can monitor and predict their evolution across different versions of software applications. This paper presents an approach for prediction of software clone evolution across different versions of software applications.

The previous work done is based on various ways of identifying cloned components in a software system [3]–[5]. But many of the papers analyzed clones in the same version of software systems. In this paper, we have analyzed the clones across different versions of software applications. The first part of our work is to identify the cloned components. The primary focus of our work is to model the evolution of cloned components using time series modeling.

If we can extract the average number of clones for each version of a particular software application, and model them, then it will be an effective approach in reducing the maintenance effort. This paper presents a time series approach for software clone evolution prediction. We have used a comparative approach for modeling the software clone evolution. First, we have modeled the software clone evolution using standard ARIMA [6] models. Then we have used nonlinear neural network model trained with ''BFGS quasi-Newton Method'' based Back-propagation [7] for modeling the clone evolution series. Finally, we have used a Multi-Objective Genetic Algorithm based Neural Network [8]–[10] for modeling the evolution of software clone components across different versions of software applications.

We have applied our approach to 31 versions of ArgoUML [11], a UML modeling tool that supports the standard UML 1.4 diagrams. ArgoUML is written in Java and hence can run on any Java platform. ArgoUML was

2169-3536 © 2017 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

developed as an open-source project and released under Eclipse Public License (EPL) 1.0. It is an open source software application, and it also won the Software Development Magazine's annual Readers' Choice Award in 2003.

We have organized the rest of the paper as follows: Section II describes a detailed description of different Time Series Modeling techniques used for clone evolution prediction Section III presents a description of Clone Detection Process implemented in this paper. Section IV describes the Data Preparation; Section V outlines the Design of Experiments, Section VI describes the Evaluation and Interpretations of Results. Section VII describes the Application of Clone Evolution Prediction Modelling. Section VIII concludes the paper and shows direction for the future work.

II. TIME SERIES MODELLING OF SOFTWARE CLONE EVOLUTION

A series of observations made sequentially in time forms a time series. It can be modeled using stochastic processes [12]. A stochastic process evolves with time and is governed by the probabilistic laws. A collection of random variables ordered in time and defined at a set of time points is called a stochastic process. A stochastic process may be discrete or continuous.

The fundamental objective of modeling a time series is prediction. Given an observed time series, we can predict its future values [13]. The prediction of future values requires designing a suitable model describing the time series accurately. The methods used to model a time series include Box-Jenkins ARIMA models, Box-Jenkins Multivariate Models, and Holt-Winters Exponential Smoothing (single, double, triple) [14].

In this paper, we have used ARIMA as well as a Neural Network model for predicting the evolution of software clone components. The neural network is first trained with ''BFGS quasi-Newton Method'' based Back-propagation [7], [15] a standard algorithm used for training of neural network. We have also used Multi-Objective Genetic Algorithm based Neural Network training algorithm [9] for improving the accuracy of the model in prediction. The goal is to have a suitable model for prediction of software clone evolution which will be helpful in reducing the software maintenance effort. The detailed description of all the models is described below.

A. ARIMA

$ARIMA(p, d, q)$ [6] is a combination of $AR(p)$, $MA(q)$ and $ARMA(p, q)$ classes. $AR(p)$ [6] is an autoregressive model of order p.

p represents the value of a series as an auto-regression of previous p values. An autoregressive model of order p abbreviated as $AR(p)$, is described in Equation 1.

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + ...\alpha_p x_{t-p} + \epsilon_t \quad (1)$$

Here, x_t is a stationary process. $\alpha_1, \alpha_2, ...\alpha_p$ are constants and $\alpha_p \neq 0$. AR (p) models the Time series x_t as a dependent value on $x_{t-1}, x_{t-2}, ...x_{t-p}$ and white noise ϵ_t. The moving average $MA(q)$ [6] is used for the white noise terms.

$$x_t = \epsilon_t + \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + ...\beta_q \epsilon_{t-q} \quad (2)$$

Here, there exist q lags in the moving average. $\beta_1, \beta_2, ...\beta_q$ ($\beta_q \neq 0$), are parameters of the moving average model. The $MA(q)$ assumes x_t as a combination of white noise terms from time points $t, t-1, t-2, ...t-q$. $ARMA(p, q)$ is the combination of $AR(p)$ and $MA(q)$ model.

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + ...\alpha_p x_{t-p} + \epsilon_t + \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + ...\beta_q \epsilon_{t-q} \quad (3)$$

Here, $\alpha_p \neq 0, \beta_q \neq 0$.

The parameters p and q represents autoregressive and moving average respectively [6]. The time series needs to be differentiated before applying $ARMA(p, q)$ model. ARIMA includes the differentiating operator d [6].

1) Model Identification: In the Model identification phase the d value has to be set. It decides the stationary (d = 0) or non-stationary (d > 0) behavior of a time series. ACF and PACF plots are plotted to find out the parameters. The identification of (p, q) is based on Akaike Information Criterion (AIC). The model with smallest AIC is chosen [6].
2) Estimation: In this phase, the coefficient α_p and β_q are estimated [6].
3) Diagnostic Checking: The diagnostic phase deals with model adequacy by plotting the residuals. The model with the smallest residual is chosen [6].

B. ARTIFICIAL NEURAL NETWORK MODELLING

MLP (Multilayer Perceptron) [16] is the frequently used artificial neural network in the field of machine learning. They are widely used in pattern recognition, text classification, etc. due to their high learning capabilities which are because of their highly complex structures as well as sophisticated training and learning functions. Many times in the field of software engineering, they have been used for the Defect prediction as well as Reliability prediction. The conventional methods of time-series predictions, i.e. Autoregressive and Moving Average, etc. are not much efficient in the predictions of the complex real-time time series data. Clone evolution prediction is a challenging task because of the complex variability of the number of clones in the different versions of the software. The MLP contains the following processing units:

1) An Input Layer
2) An Output Layer
3) One or more Hidden Layers

Each layer consists of several numbers of neurons based on the problem specification and also designed by the user. Each layer receives inputs from the outputs generated by the previous layer and also produces the output through an activation function (may be linear or nonlinear as per requirements). The output is passed to the next layer in the network.

1) BACK PROPAGATION LEARNING

MLP is mostly used for nonlinear learning. It is trained using supervised learning approach. The models adjust the network weights on its inputs and the internal nodes iteratively. The goal is to minimize the errors between actual and predicted values. The BFGS quasi-Newton method [7] is an advanced algorithm used for neural network training. For speeding up the computing process, the data needs to be normalized before being trained by a neural network. Different transfer functions are used to catch the linear and nonlinear patterns in the data. As the real-time bug number data shows nonlinear behavior, the nonlinear transfer functions are used for the hidden layer.

Let X be the input vector comprising of the features of a particular training example.

Let the output vector of the hidden layers be $Y_1, Y_2, Y_3, \ldots\ldots, Y_m$, where m = number of hidden layers.

The final output Y is obtained through the layer-by-layer transfer of the output.

The transfer function used is the logarithmic sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$
$$Y_1 = f(b_1 + w_1^T X)$$
$$Y_2 = f(b_2 + w_2^T Y_1)$$
$$\ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots$$
$$Y = f(b_m + w_m^T Y_m)$$

Where w_i are the weight vectors of the neurons of i-th layer and b_i is the vector of the bias terms for each neuron of ith layer.

2) MULTI-OBJECTIVE GENETIC ALGORITHM (MOGA-NN)

Neural networks training is performed to minimize the training error corresponding to the given training data. The optimization technique used is generally Back-propagation [7]. Genetic Algorithm [Zhou201132] is used to minimize the cost function and to improve the accuracy. In our context, as we have two objective functions as a cost function, we will use Multi-Objective Genetic Algorithm [8] to optimize them.

A genetic algorithm [17] is one of the most efficient methods of both constrained and non-constrained optimizations. It mimics the natural process of selection based on the fitness of the population. The genetic algorithm has following steps:

1) Generate Initial Population: A random population of n chromosomes is generated. (Each chromosome represents suitable solution for the problem)
2) Evaluation of fitness function: The fitness f(x) of each chromosome x in the population is Evaluated.
3) Generate a New Population: Create a new population by eliminating the weak population (Population which does not satisfy fitness criteria).
4) Selection of Parents: Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
5) Perform Crossover Operation: With a particular crossover probability, the crossover operation is applied to generate new offspring.
6) Perform Mutation Operation: With a particular mutation probability, the mutation operation is conducted to make changes at different locations of new offspring. Place new offspring in the new population
7) Replace the old Population by a new one: Now the Genetic algorithm has to be applied to the new population.
8) Stopping Criteria: If the stopping condition is satisfied, stop the algorithm. The best solution from the current population is returned, else
9) Loop: Go to step 2

The Multi-Objective Genetic Algorithm [17] is the application of the Genetic Algorithm to optimize multiple objectives. It gives a set of optimal values $x = x_1, x_2, \ldots x_l \in R$ which minimizes a set of given objective functions subject to a given set of constraints (if any). Basically, instead of point prediction through ANNs, we are here going to have prediction intervals for a sample as they can handle the uncertainty in the model parameters and also noise in the input data [18].

3) PREDICTION INTERVALS

Prediction Interval (PI) [19] is nothing but the upper bound and the lower bound of the prediction. For a given dataset $X = \{(x_i, y_i) \mid i = 1, 2, \ldots n\}$, Let the prediction interval of (x_i, y_i) *be* $[L(x_i), U(x_i)]$. We define the Mean Prediction Interval Width (MPIW) and Prediction Interval Coverage Probability (PICP) for X as:

$$MPIW = \frac{1}{n}\sum_{i=1}^{n}(U(x_i) - L(x_i))$$
$$PICP = \frac{1}{n}\sum_{i=1}^{n} c_i,$$
$$where \quad c_i = \begin{cases} 1, & if \quad y_i \in [L(x_i), U(x_i)] \\ 0, & otherwise \end{cases}$$

For a better interval prediction, we will optimize the parameters PICP [18], [20], [21] to be maximum and MPIW [20], [21] to be minimum. We also see that as MPIW decreases, PICP increases. Hence we have a competitive multi-objective optimization problem:

$$Minimize \quad NMPIW(X)$$
$$and\, 1 - PICP(X)[19]$$
$$Simultaneously$$
$$Such\ that, NMPIW \geq 0 \ \ and \ \ 0 \leq PICP(X) \leq 1$$

We have taken normalized $MPIW(NMPIW)$ [19] instead of $MPIW$,

$$where \quad NMPIW = \frac{MPIW}{y_{max} - y_{min}} \ and$$

We choose to minimize (1-PICP) instead of maximizing of $PICP$ because both are equivalent.

The optimal set x is known as a Pareto-optimal set. It is a set of solutions, satisfying the objective functions at accepted levels without being dominated by any other solution.

In other words, if we have to minimize N objective functions, $f_1(x), f_2(x), \ldots f_n(x)$, subject to some given constraints, Let X be the solution space of feasible solutions, then $x_1 \in X$ is said to dominate $x_2 \in X$

$$\begin{aligned} &iff \quad f_i(x_1) \leq f_i(x_2) \\ &\quad for\ all\ i \in [1, N]\ and \\ &\quad\quad f_j(x_1) < f_j(x_2)\ for\ some\ j \in [1, N]. \end{aligned}$$

Hence, for x being a Pareto optimal set, it should not be dominated by any other solution in the solution space.

4) NSGA-II [8]

It is the most common form of MOGA. In this algorithm, after each generation, on the new population non-dominance sorting algorithm [8] is applied to sort the chromosomes in the order of dominance. The chromosomes are selected by crowding distance by using binary tournament selection. The crossover and mutation operation is performed to produce the next generation, which also becomes a part of the population. As soon as the maximum number of generations is reached, the first Pareto front of the sorted population is returned, and the corresponding set is the Pareto optimal set [19].

Here we have presented a brief description of the implementation of MOGA in neural networks:

1) Initialize a feed forward neural network N *where* $N(x_i)$ represents the predicted value of the training sample $xi \in X$.
2) Train the neural network with MOGA as the optimization function for minimizing the cost functions, i.e. $NMPIW(X)$ *and* $1 - PICP(X)$.
3) The Pareto optimal set of weights and biases, P (Pareto Set) obtained after the neural network training, is used to find the most optimal solution by setting each set of weights $(w_i, b_i) \in P$ and calculating the NRMSE for the training dataset X.
4) The solution yielding the minimum training NRMSE is the most optimal set of weights, which can be now used to train the neural network again and apply on the test dataset.

The Flow Diagram for MOGA-NN implementation is given in figure 1 on page 11844.

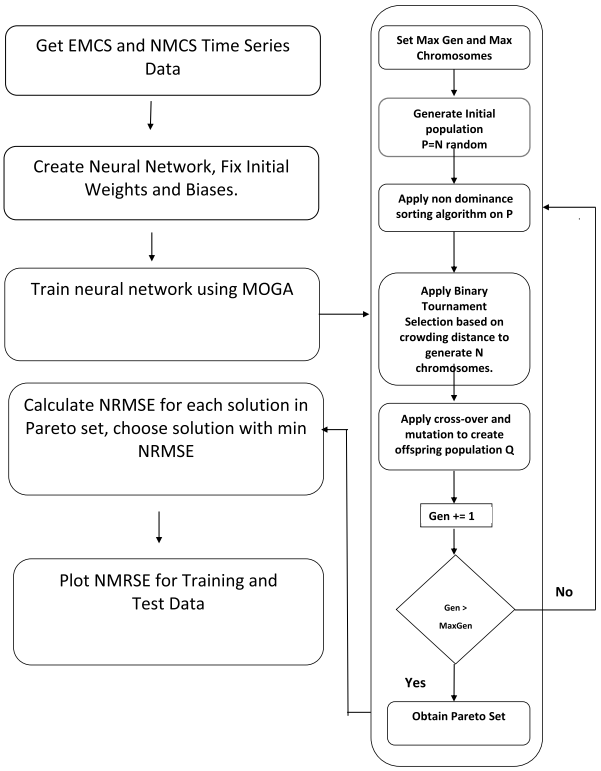


FIGURE 1. Flow diagram representation for MOGA-NN.

III. SOFTWARE CLONE DETECTION

As software evolves, new code fragments are added, deleted or modified to fulfill the desired requirements. These duplicated, or slightly modified code fragments are called cloned components and focus is on detection of these cloned components. As there is no prior knowledge about which code fragments may be repeated, the detector has to compare every possible fragment with every other possible fragment. The comparison may become intractable from a computational viewpoint. Therefore several measures are used to reduce the domain of comparison before performing the actual comparisons. A typical clone detection process involves following steps.

A. PRE-PROCESSING

Pre-processing partitions the source code and determines the domain of comparison. Three primary objectives exist in this phase [3], [5], [22].

1) FILTERING THE CODE

This involves eliminating code elements that are not required in comparison. This step is necessary for situations when the comparison tool is not language independent. E.g.: SQL statements embedded in JAVA code. Also, generated sources, such as table initialization can be removed during this phase.

2) DETERMINE SOURCE UNITS

The source code is partitioned into disjoint units which are directly involved in clone relations with granularity levels varying from Statements, Blocks, and Functions, etc. to Files.

3) DETERMINE COMPARISON GRANULARITY

This involves to set up the minimum units for comparison, or to fix the granularity parameter for comparison. This parameter affects the running time of the analysis.

B. TRANSFORMATION

Transformation converts the given pre-processed code into an intermediate representation for the subsequent clone

extraction process. Normalization additionally removes whitespaces and comments. Normalizing transformation is also supported by some tools [2].

C. CLONE EXTRACTION

Source code transformation is also referred as extraction. It involves one or more of the following steps:

Tokenization: In token-based approach, using lexical analysis, the source code is divided into tokens. These tokens help in the formation of sequences, which are compared. All the whitespaces and comments are removed [23].

Parsing: In syntactic approaches, an abstract syntax tree [AST], possibly annotated, and is built from the source code [23].

D. MATCH DETECTION

At the final stage, the comparison algorithm works on the transformed units to produce clone sets. Adjacent comparison units may be aggregated to form larger units. The granularity for comparison can be fixed or set as a parameter. The output is a set of clone pairs with the location of the clone fragments in the source code.

E. CLONE DETECTION

Abstract syntax tree based clone detection technique is used in this paper with the help of CloneDr, a clone detection tool [23]. After processing, Exact-match clone sets and Near-miss clone sets were obtained.

1) Exact-match Clone Sets (EMCS): It is the count of cases in which some code block has been cloned without any changes [23].
2) Near-miss Clone Sets (NMCS): It is the count of abstractions with parameters for which there are multiple clone instances [23].

Some Input Parameters used for Clone Estimation

1) Similarity threshold is a real value between 0 and 1 which represents the ratio of the volume of identical code in a clone, to the actual total code in the clone.
2) Volume is computed regarding Abstract Syntax Tree (AST) nodes [5], but can be effectively understood regarding Source Lines of Code (SLOC). This threshold controls how similar two blocks of code must be to be considered as clones [24].
3) Max Clone Parameter helps the tool to determine a limit on dissimilarity between near-miss clones, thereby restricting the number of parameters, which each abstraction is allowed to have [24].
4) Minimum Clone Size helps control how big a piece of code must be to be considered a clone. This value c is used to determine an AST node count threshold,computed as:
$c \times \frac{A}{L}$,
where A is the total number of AST nodes and L is the number of physical lines in the complete software [23]

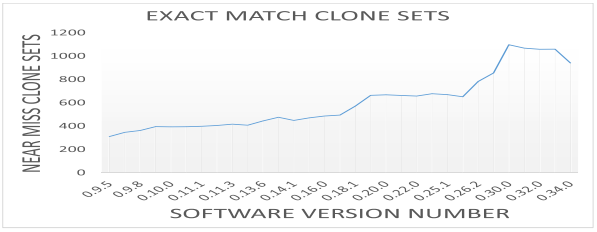


FIGURE 2. Time series representation for exact match clone sets for ArgoUML.

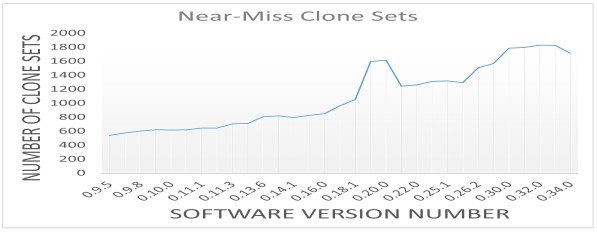


FIGURE 3. Time series representation for near miss clone sets for ArgoUML.

Figure 2 and Figure 3 shows the time series representation for EMCS and NMCS respectively for ArgoUML [11].

IV. DATA PREPARATION

A. APPLIED DATASET

We have implemented our experiments on 31 versions of ArgoUML [11] ranging from version 0.9.5 to the latest stable release 0.34.0. ArgoUML is a UML modeling tool that supports the standard UML 1.4 diagrams. ArgoUML is written in Java and hence can run on any Java platform. ArgoUML was developed as an open-source project and released under Eclipse Public License (EPL) 1.0. In 2003, ArgoUML won the Software Development Magazine's annual Readers' Choice Award in the "Design and Analysis Tools" category.

ArgoUML supports code generation for $C, C++$, *Java*, *C#*, *PHP4* *and* *PHP5* and it also supports reverse engineering from Java. It also has XML support. The flexible feature of ArgoUML is that it allows you to export your diagrams in different file formats such as GIF, PNG, PS, EPS, PGML, and SVG.

The interactive user interface of ArgoUML provides its users the ability to zoom and edit diagrams. ArgoUML is now available in ten different languages to support a variety of population. It also has support for OCL. Another interesting feature of ArgoUML is the presence of checklist for every component of a model.

The designing process of ArgoUML is user-friendly because of the assistance features provided at each step of the process.

The screen of ArgoUML is divided into four sections. The explorer section shows the relationship between diagrams and design items. To do section provides a list of pending tasks. The user creates and design diagrams in the main window. In the details section, the user defines the diagrams and link them with elements.

B. DATA PARTITION

The entire dataset prepared by the clone number series is partitioned into a training set and a testing set. It is essential in finding the best fit model with the data pattern for both trained and test data. The model giving more accuracy on the test data is preferably selected. The size of the training dataset and test dataset are taken as 25 and 6 respectively for all the models, to have a consistent comparison among the three models.

V. DESIGN OF EXPERIMENTS

In the first phase, we got the cloned components by AST based clone detector, the CloneDr [4]. For the ArgoUML software application, the result of the clone detection phases is the exact match and near miss clone components. We got time series of size 31 for each type of cloned components. The next step is to model them using suitable time series modeling. First, we have applied the ARIMA [13] to model each type of clone time series. Then we have used the neural network with Back Propagation and MOGA respectively for modeling of the clone evolution. The description of the implementation of all the models is given in this section.

A. ARIMA MODELLING

AS discussed in the previous section; The ARIMA [13] model consists of AR (p), MA (q) and a differentiating (d) operator. ARIMA is a good predictor of stationary time series. ACF and PACF plots are used to for determination of stationary behavior of the series. If the ACF decays slowly, the behavior of the series is non-stationary and if ACF decays instantly, the behavior of the time series is stationary. For the EMCS and NMCS, we plot the ACF and PACF to find out the p, d and q parameters. The representation of ARIMA model is given in Figure 6a.

B. NEURAL NETWORK MODELLING

1) BACK PROPAGATION BASED LEARNING

In this paper, we have also implemented a multi-layer neural network having one input layer, one hidden layer, and one output layer. From the ARIMA model we obtained the autoregressive parameter(p) for both EMCS and NMCS value as two; i.e. the current value is dependent upon 2 past values of the series. Hence the input layer will contain two neurons for the two lags as features, the hidden layer can include a custom number of neurons which we have optimally chosen by error-and-trial as 10, and finally the output layer includes single neuron denoting the NRMSE value. The diagrammatic representation of an implementation of Back Propagation Based Neural Network model is given in Figure 6b.

2) MULTI-OBJECTIVE GENETIC ALGORITHM BASED NEURAL NETWORK LEARNING

Neural networks training is performed to minimize the error corresponding to the given training data. The optimization technique used is generally ''Back-Propagation'' using gradient descent [7]. Genetic Algorithm [Zhou201132] is used

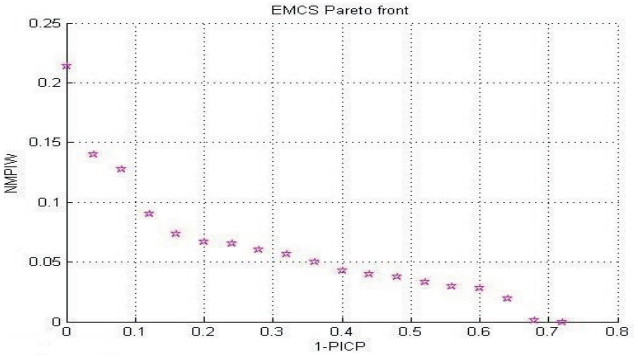


FIGURE 4. Pareto plot between X-AXIS:(1-PICP) and Y-AXIS: NMPIW(EMCS).

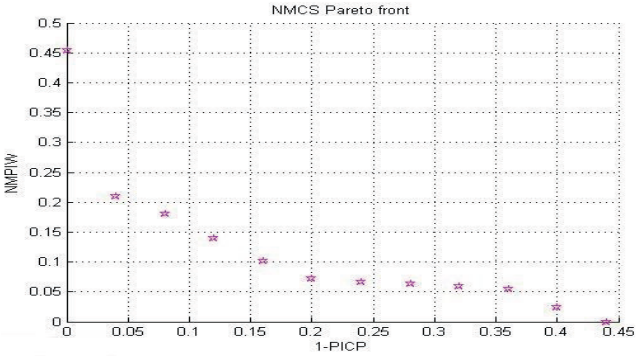


FIGURE 5. Pareto plot between X-AXIS:(1-PICP) and Y-AXIS: NMPIW(NMCS).

to minimize the cost function and to improve the accuracy. In our context, as we have two objective functions as a cost function. We have used Multi-Objective Genetic Algorithm to optimize the error. The diagrammatic representation of an implementation of Back Propagation Based Neural Network model is given in Figure 6c. Here is a brief description of the implementation of MOGA in neural networks.

Steps for Implementation MOGA in Neural Network:

1) The ArgoUML [11] dataset for 31 versions was divided into training dataset with 25 samples and test dataset with remaining 6 samples.
2) The training data was fed into an artificial neural network consisting of an input layer, a hidden layer, and output layer composed of two neurons each.
3) The neural network was trained to optimize the prediction intervals for the training data, through minimizing NMPIW and maximizing PICP. The Pareto Front for EMCS and NMCS after applying MOGA-NN is presented in Figure 4 and 5 respectively.
4) The optimal set of weights and bias terms, also known as a Pareto-optimal set, is used to find the most optimal solution by setting them one by one and getting the most optimal weights and bias which gives minimum NRMSE.
5) The trained model is applied to the test data to get the test accuracy.

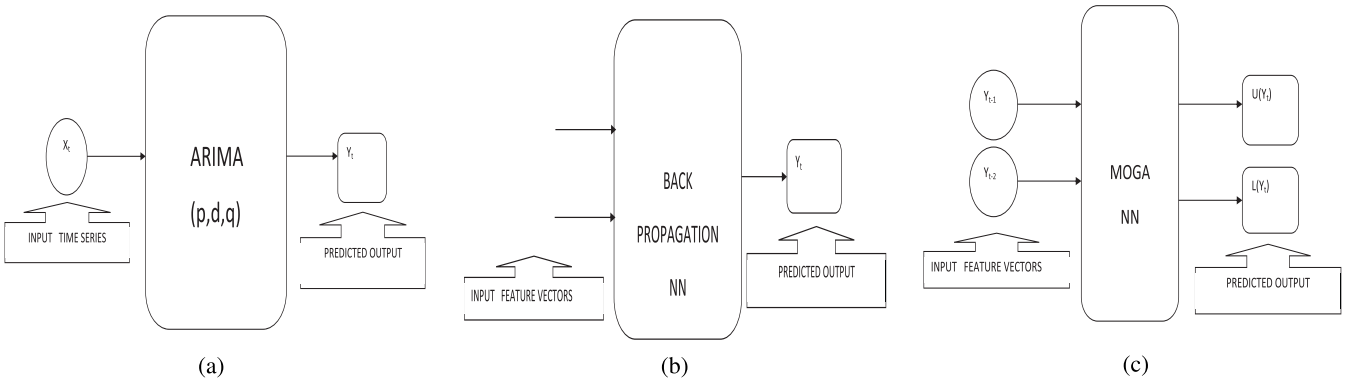


FIGURE 6. Representation of modelling techniques. (a) ARIMA model representation. (b) Back-propagation based neural network. (c) MOGA-based neural network modelling for EMCS and NMCS.

TABLE 1. Parameters description.

ARIMA	
p	Autoregressive Parameter
d	Differencing Parameter
q	Moving Average Parameter
H	Forecast Horizon
NN(BackPropagation	
I	Number of Input Neuron
O	Number of Output Neuron
T_r	Transfer Function
T_n	Training Function
Reg.P	Regularization Parameter
λ	Learning Rate
NN(MOGA)	
I	Number of Input Neuron
O	Number of Output Neuron
T_r	Transfer Function
C_p	Crossover Probability
M_p	Mutation Probability
Max_G	Maximum No. of generation
n_p	Number of Chromosome

The description of parameters for all the three models is given in Table 1 and the value of all the parameters is given in Table 2.

VI. EVALUATION AND INTERPRETATION

To evaluate the Prediction result, we have standard assessment methodology; the normalized root mean square error (NRMSE) [25]. The NRMSE can be defined as:

$$NRMSE = \frac{1}{y_{max} - y_{min}} \left(\sqrt{\frac{(\sum_{t=1}^{n_p} (y_t - \hat{y}_t)}{n_p}} \right)$$

For ARIMA [6] and Back Propagation [7] based Neural Network modeling NRMSE is directly calculated from the point estimates for training and test data. For MOGA - NN [19] approach, the bounds of PIs, are computed first. NRMSE is calculated by taking the mean of the upper bound and lower bound of each training sample. The models are evaluated both on the trained and test data. The model with minimum NRMSE on test data is selected.

In this section, we give a detailed presentation of NRMSE value by all the three models.

A. ARIMA MODEL EVALUATION

As described in the previous section, the most appropriate ARIMA model that gives the best fit for the clone evolution series is ARIMA (2, 1, 2). The model is applied to both EMCS, and NMCS clone evolution series and the forecast value is also predicted for the next six values in the series. The Training error and Prediction error for both EMCS and NMCS are presented in Table 3. Figure 7a and Figure 8a represent the diagrammatic representation of original vs. ARIMA predicted series for EMCS and NMCS respectively.

From the Table 3, we interpreted that ARIMA model is a poor predictor for the test data for both EMCS and NMCS series respectively. As we see from the graph, there is a significant deviation from the original series from 25 to 31 which are the forecasted values by ARIMA model. This result shows the poor predictive capacity of Linear ARIMA model for clone evolution data which contain a mixture of linear and nonlinear components.

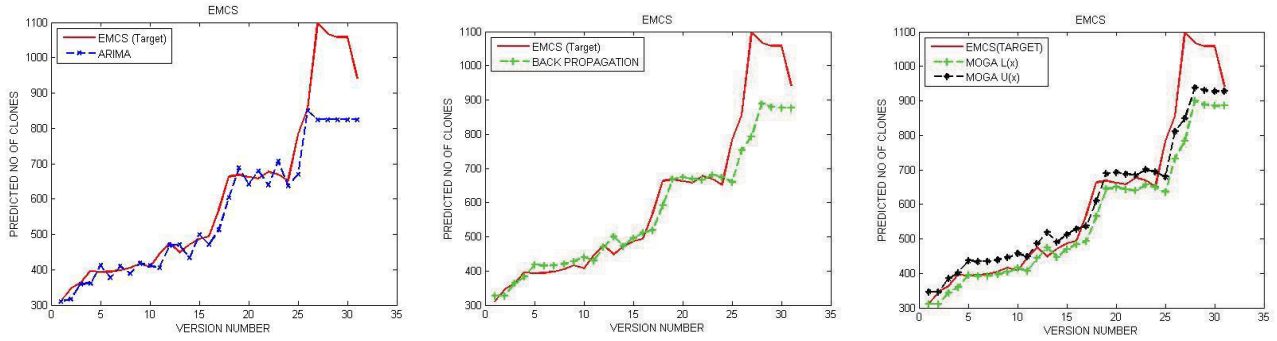
B. BACK PROPAGATION BASED NN EVALUATION

As discussed in the previous section, we have a multi-layer perceptron having a single input, hidden and an output layer. We have also mentioned the number of neurons in each layer in the previous section. The model is trained with BFGS quasi-Newton method, and the NRMSE is calculated for the train and test data. The model is validated against both EMCS and NMCS respectively. The NRMSE as given by the model is presented in Table 4. Figure 7b and Figure 8b represent the diagrammatic representation of original vs. MLP predicted series for EMCS and NMCS respectively.

From the Table 4, we found an improvement of prediction accuracy for MLP modeling. We also see that the NRMSE for the test data for both EMCS and NMCS is lower than the ARIMA modeling. However, there is still a deviation from the original series in the region 25 -31 which needs to be minimized to make this model more reliable. Though back propagation is a good predictor of nonlinear data,

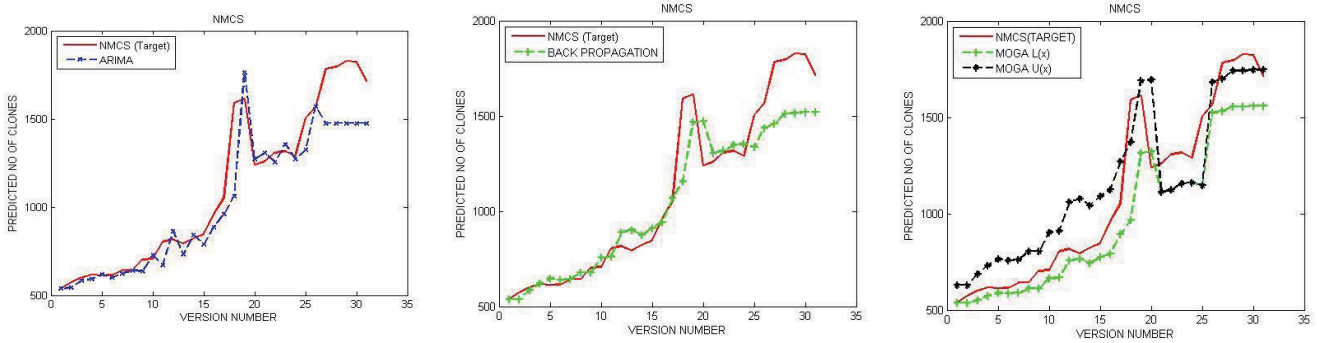
TABLE 2. Value of parameters for different models.

	ARIMA				NN (BACK Propagation)						NN (MOGA)						
Parameters	p	d	q	H	I	O	Tr	Tn	Reg.P	λ	I	O	Tr	C_p	M_p	Max_G	N_p
EMCS	2	1	2	6	2	1	Tansig	$BFGS-QN$	0.01	0.15	2	2	Tansig	0.8	0.06	100	25
NMCS	2	1	2	6	2	1	Tansig	$BFGS-QN$	0.01	0.15	2	2	Tansig	0.8	0.06	100	25



(a) (b) (c)

FIGURE 7. Plotting of actual vs. predicted series (EMCS). (a) Actual vs. ARIMA predicted series (EMCS). (b) Actual vs. back propagation based NN predicted series (EMCS). (c) Actual vs. MOGA-based NN predicted series (EMCS).



(a) (b) (c)

FIGURE 8. Plotting of actual vs. predicted series (NMCS). (a) Actual vs. ARIMA predicted series (NMCS). (b) Actual vs. back propagation based NN predicted series (NMCS). (c) Actual vs. MOGA-based NN predicted series (NMCS).

TABLE 3. ARIMA result.

	ARIMA-TRAIN	ARIMA-Test
EMCS	0.045	0.263
NMCS	0.097	0.27

TABLE 4. NN- Back Propagation result.

	NN-BackPropagation-TRAIN	NN-BackPropagation-Test
EMCS	0.0445	0.234
NMCS	0.0894	0.207

simultaneously it also suffers from local optima problem. We need to further optimize the error (NRMSE) for more reliable and robust model for clone evolution prediction.

C. MOGA-NN EVALUATION

The detailed procedure for implementation of MOGA-NN is already described in the previous section. Here the bounds of PIs, are calculated first. NRMSE is calculated by taking the mean of upper bound $U(X)$ and lower bound $L(X)$ of each training sample x.

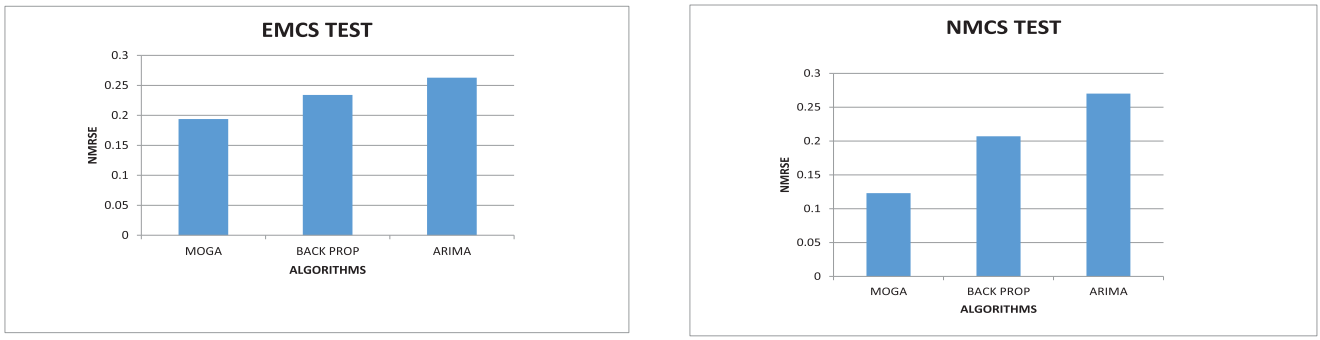
The NRMSE as given by the MOGA-NN model is presented in Table 5. Figure 7c and Figure 8c represents the diagrammatic representation of original vs. MOGA NNpredicted series for EMCS and NMCS respectively.

TABLE 5. NN- MOGA result.

	NN-MOGA-TRAIN	NN-MOGA-Test
EMCS	0.043	0.194
NMCS	0.094	0.123

TABLE 6. Comparison of different models.

Comparison of Properties of Different Models		ARIMA	NN-BackPropagation	NN-MOGA
1	Generalization Ability	Poor	Good	Optimal
2	Complexity of Algorithm	Simple Auto-Regression and Moving Average ensemble	Simple BackPropagarion-Trained Neural Network	MOGA-Trained Neural Network
3	Parameter Setting	Parameters setting done by ACF and PACF plots	Parameter Tuning required for Learning rate, Regularization Parameter and Training Function	Parameters tuning required for number of chromosomes, Max generations and crossover and mutation functions
4	Training Function Characteristic	Auto Regression, Moving Average	BFGS quasi-Newton method	Multiobjective Genetic Algorithm NSGA-II
5	Computation Load	Negligible	Low for both Training and Test data	High for the Training data; Low for Test data
6	Prediction Accuracy	Poor	Average	Optimal
7	Suitability (For Clone Evolution Prediction)	Not suitable	Medium Suitability	Most Suitable



(a) (b)

FIGURE 9. NRMSE comparison between three models. (a) EMCS NRMSE test data. (b) NMCS NRMSE test data.

From the Table 5, we see that MOGA-NN improves not only the accuracy but also a most reliable predictor of the clone evolution. We found from Figure 7c and 8c respectively that Interval estimates are robust method of predicting the series as it completely captures the series within the intervals except for some points. Point estimates sometimes are affected by the uncertainties of the model parameters. We also see that the intervals give a clear picture of the prediction. From the table, we also found that the NRMSE value for the MOGA-NN model outperforms both ARIMA and Backpropagation model for the test data for both EMCS and NMCS. Figure 9a and Figure 9b present a bar chart representation of NRMSE for both EMCS and NMCS for test data only.

As discussed in the previous section the model which gives more accuracy (Minimum NRMSE) has to be selected for prediction of clone evolution. From the Result, we interpreted that MOGA - NN is a most suitable model for clone evolution prediction as it is more accurate, robust and reliable.

ADVANTAGES OF MOGA-NN OVER OTHER MODELS

- It also gives us a broad scope of decision-making by providing a Pareto set of optimal solutions which provide valuable information about the underlying problem.
- Also, NSGA-II does not require derivative calculations or calculation of Jacobian or Hessian matrices.
- MOGA does not get stuck at local optimal solutions like the gradient descent techniques.

In table 6 we have compared the three different techniques based on seven criteria in details.

VII. APPLICATION OF THE MODEL

The work involves identification of cloned components and also the prediction of clone evolution content in an open source software applications. The identification of duplicated and nearly duplicated code is also immensely helpful in the field of bug prediction. It is also useful for reducing the Corrective Maintenance [26] and Preventive Maintenance [27]

which involves modification of code content to solve and prevent problems in the software respectively. Because if we can identify and detect the cloned areas, the defect in all the similar code fragments can be resolved at once.

The software clone evolution prediction is immensely helpful in Perfective Maintenance [27], [28] and Adaptive Maintenance [27], [28] because the effort required to evolve a software is also dependent on the amounts of cloned contents in the software [28]. Clone evolution prediction is also helpful in the validation of many software evolution hyphotheses [29]. The customer also can evaluate the evolution of clone content for taking decisions on purchasing new versions of software applications.

VIII. CONCLUSION AND FUTURE WORK

The software testing and maintenance are most expensive part in the software engineering [30]. In this paper, we have discussed on a method for proper monitoring and predicting the evolution of clone numbers across different versions of the open source software applications. It is helpful in reducing the effort for software maintenance [26], [31], [32]. It is also useful in validating some software evolution hypotheses [29].

In this paper, we also give a comparative analysis of the predictive performance between ARIMA and two different types of Neural Network Modeling. The result confirms the MOGA-NN model as a best predicting model for both types of clone number series. This paper also confirms the poor performance of ARIMA model in predicting the non-linear patterns in data.

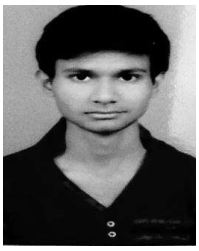
In the future, we can have a detailed analysis of the relationship of software metrics and software clones. It can give a clearer picture of clones in software. We can also model the increasing and decreasing temporal patterns of the software clone evolution using advanced modeling techniques. We can also predict whether code-refactoring is required in case the code fragments smell bad i.e. the error-prone code fragments in the software.

REFERENCES

[1] U. Raja *et al.*, "Temporal patterns of software evolution defects: A comparative analysis of open source and closed source projects," *J. Softw. Eng. Appl.*, vol. 4, no. 8, p. 497, 2011.
[2] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," in *Proc. 4th Int. Conf. Adv. Comput. Commun. Technol.*, 2014, pp. 189–196.
[3] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, 2009.
[4] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 577–591, Sep. 2007.
[5] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1165–1199, 2013.
[6] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "Time series analysis for bug number prediction," in *Proc. 2nd Int. Conf. Softw. Eng. Data Mining (SEDM)*, 2010, pp. 589–596.
[7] A. Ghosh and M. Chakraborty, "Hybrid optimized back propagation learning algorithm for multi-layer perceptron," *Int. J. Comput. Appl.*, vol. 60, no. 13, Dec. 2012.
[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[9] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Safety*, vol. 91, no. 9, pp. 992–1007, Sep. 2006.
[10] C. C. Coello, G. B. Lamont, and D. A. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. New York, NY, USA: Springer, Sep. 2007.
[11] *Argouml Database ONLINE*, accessed on Feb. 7, 2015. [Online]. Available: http://argouml.tigris.org/source/browse/argouml/trunk/src/
[12] T. Fathima and V. Jothiprakash, "Behavioural analysis of a time series—A chaotic approach," *Sadhana*, vol. 39, no. 3, pp. 659–676, 2014.
[13] N. Tran and D. A. Reed, "Arima time series modeling and forecasting for adaptive i/o prefetching," in *Proc. 15th Int. Conf. Supercomput.*, 2001, pp. 473–485. [Online]. Available: http://doi.acm.org/10.1145/377792.377905
[14] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and its Application With R Examples*. New York, NY, USA: Springer, 2006.
[15] E. Castillo, B. Guijarro-Berdiñas, O. Fontenla-Romero, and A. Alonso-Betanzos, "A very fast learning method for neural networks based on sensitivity analysis," *J. Mach. Learn. Res.*, vol. 7, pp. 1159–1182, Jul. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248589
[16] D. Ruta and B. Gabrys, "Neural network ensembles for time series prediction," in *Proc. Int. Joint Conf. Neural Netw.*, 2007, pp. 1204–1209.
[17] K. Fung, C. Kwong, K. W. Siu, and K. Yu, "A multi-objective genetic algorithm approach to rule mining for affective product design," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7411–7419, 2012.
[18] D. L. Shrestha and D. P. Solomatine, "Machine learning approaches for estimation of prediction interval for the model output," *Neural Netw.*, vol. 19, no. 2, pp. 225–235, 2006.
[19] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "Lower upper bound estimation method for construction of neural network-based prediction intervals," *IEEE Trans. Neural Netw.*, vol. 22, no. 3, pp. 337–346, Mar. 2011.
[20] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics Intell. Lab. Syst.*, vol. 39, no. 1, pp. 43–62, 1997.
[21] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "Comprehensive review of neural network-based prediction intervals and new advances," *IEEE Trans. Neural Netw.*, vol. 22, no. 9, pp. 1341–1356, Sep. 2011.
[22] N. Gode and R. Koschke, "Incremental clone detection," in *Proc. 13th Eur. Conf. Softw. Maintenance Reeng. (CSMR)*, 2009, pp. 219–228.
[23] I. D. Baxter, A. Yahin, L. Moura, and M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proc., Int. Conf. Softw. Maintenance*, 1998, pp. 368–377.
[24] R. Smith and S. Horwitz, "Detecting and measuring similarity in code clones," in *Proc. Int. Workshop Softw. Clones (IWSC)*, 2009.
[25] H. Liu, H.-Q. Tian, D.-F. Pan, and Y.-F. Li, "Forecasting models for wind speed using wavelet, wavelet packet, time series and artificial neural networks," *Appl. Energy*, vol. 107, pp. 191–208, Jul. 2013.
[26] *Software Engineering–Software Life Cycle Processes–Maintenance*, ISO Standard 14764, 2006.
[27] H. M. Sneed, "Offering software maintenance as an offshore service," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Oct. 2008, pp. 1–5.
[28] A. April and A. Abran, *Software Maintenance Management: Evaluation and Continuous Improvement*, vol. 67. Hoboken, NJ, USA: Wiley, 2012.
[29] G. Xie, J. Chen, and I. Neamtiu, "Towards a better understanding of software evolution: An empirical study on open source software," in *Proc. ICSM*, vol. 9. 2009, pp. 51–60.
[30] T. M. Pigoski, *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. Hoboken, NJ, USA: Wiley, 1996.
[31] "Systems and software engineering-vocabulary," Inst. Elect. Electron. Eng., Piscataway, NJ, USA, Tech. Rep. 247652010, 2010.
[32] P. Grubb and A. A. Takang, *Software Maintenance: Concepts and Practice*. Singapore: World Sci., 2003.

JAYADEEP PATI received the M.Tech. degree from the National Institute of Technology, Rourkela, India, in 2012, and the B.Tech. degree from the Institute of Technical Education and Research, Bhubaneswar, India, in 2010. He is currently a Research Scholar with the Department of Computer Science and Engineering, IIT Varanasi, Varanasi, India. His research interests include machine learning, software engineering.

BABLOO KUMAR is currently pursuing the B.Tech. degree with the Department of Computer Science and Engineering, IIT (BHU) Varanasi, Varanasi, India. His research interests include neural networks, data mining, and machine learning.



DEVESH MANJHI is currently pursuing the B.Tech. degree with the Department of Computer Science and Engineering, IIT (BHU) Varanasi, Varanasi, India. His research interests include artificial intelligence, sequence mining, and machine learning.

K K SHUKLA is currently a Professor with the Department of Computer Sciences and Engineering, IIT Varanasi, Varanasi, India. His current research interests include artificial intelligence, neural networks, and data mining. He is also a Reviewer for several peer-reviewed journals.