International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 11 6360 - 6365

An Enhanced Bayesian Decision Tree Model for Defect Detection on Complex SDLC Defect Data

Nageswara Rao Moparthi
Research Scholar,
Dept.of Computer Science & Technology,
Sri Krishnadevaraya University
Ananthapuram,India

Dr. N. Geethanjili
Associated professor,
Dept.of Computer Science & Technology,
Sri Krishnadevaraya University
Ananthapuram,India

***Abstract*—** In this paper, we explore the multi-defect prediction model on complex metric data using hybrid Bayesian network.Traditional software metrics are used to estimate the effect of defects for decision making. Extensive study has been carried out to find the defect patterns using one or two software phase metrics.However, the effect of traditional models is influenced by redundant and irrelevant features.Also, as the number of software metrics increases, then the relationship between the new metrics with the traditional metric becomes too complex for decision making. In this proposed work, a preprocessed based hybrid Bayesian network was implemented to handle large number of metrics for multi-defect decision patterns. Experimental results show that proposed model has high precision and low false positive rate compared to traditional Bayesian models.

Keywords—*Hybrid Model, Bayesian Model, Decision patterns,Defect data.*

*****

I. INTRODUCTION

The defect is a flaw in the software program which can cause it to fail to perform its functions. Defect prediction provides an optimized way to find the vulnerabilities in the SDLC phases which occurs due to manual or automatic errors.. As the dependency of software programs increasing, software quality is becoming more and more essential in present era. Software defects such as failures and faults may affect the quality of software which leads to customer dissatisfaction.Due to the increasing of software constraints and modular complexity , it is too difficult to produce a quality end product. Defects in software may cause loss of money and time, so it is necessary to predict bugs in advance for successful quality products and decision makers. As a result, these bug reports present in various bug tracking frameworks contains detailed information about the bugs along with the severity level[1-3]. Generally faulty constraints that causes incorrect outputs are represented as software bugs. These constraints can be defined as a set of features which can be used to find the bugs.These features influence the effectiveness of the bug prediction model. Various types of classification and feature selection models have been applied for software defect detection, including decision trees, multiple regression, neural networks, svm and naïve Bayes. However, these models have failed to select the relevant defects for appropriate classifier. The performance of software defect detection also decreases due to the noise and large number defect features [4] [5].

The basic limitations of these traditional models are :

1) Unable to find the new patterns to the dynamic features.
2) Fail to load the metric data with a large number of instances.
3) The requirement specification of the project may be wrong either due to missing features or values and contradictory requirements. It may be not well documented or too complex to analyze.
4) Metrics are not qualified enough for the project[6].
5) The software may not be tested enough or some bugs might not be fixed during the testing time.
6) Bayesian network has the capability to find node prediction and its relationship to other nodes in the network. In the software development process, bugs and their dependencies are too complex to predict due to uncertain factors that lead to defects.

A Bayesian network is a directed acyclic graph with E edges and V vertices. The set of variables in the Bayesian network represents the joint probability distribution values, and each vertex represents the variable and an edge represents the association between the vertices. Let $V = \{V_1, V_2 \ldots V_n\}$ be the discrete or continuous random variables used in the Bayesian network for defect prediction model. The probability computation of V_i is shown as $P(V_i / a_x)$ where a_x represents the parent nodes of V_i .Then the joint probability distribution of X can be given as

$$\text{Prob}(V) = \text{Prob}(V_1, V_2 \ldots V_n)$$
$$\text{Prob}(V_1 / V_2, \ldots V_n).\text{Prob}(V_2 / V_3, \ldots V_n) \ldots \text{Prob}(V_{n-1} / V_n)\,\text{Prob}(V_n)$$

$$= \prod_{i=1}^{n} \text{Prob}(V_i / V_{i+1}, \ldots V_n)$$

Feature selection is a process of selecting a relevant attribute subset of a large number of defect attributes. Feature selection can be categorized as feature ranking and subset selection. Defect feature ranking is evaluated on individual metrics and ranks attributes according to their ranking measure. Feature subset selection is used to select a subset of the features of the original attributes set with high predictive measures.

The rest of the paper is summarized as follows. The related work of the different defect prediction models and feature selection models in software defects are discussed in Section II. In section III, we proposed a new filter based hybrid Bayesian network model for defect prediction . In Section IV, experimental results are evaluated on different software defects datasets and finally, Section V describes about conclusion and future scope.

II. Related Work

[1][2] formulated the defect prediction models to find the stochastic process in terms of defect variables and find the interval between the variable rate. They used non-homogeneous poison process to formulate the number of defects found during the defect dependency test. For each defect find the poison process, P(t), the probability of finding k defects by the time t and it is expressed in terms of the Poisson distribution with mean m (t) as

$$\text{Prob}\{P(t)=k\}=m(t)^n.\ e^{-mt}/n!$$

The exponential model is used to find the defect distribution in the testing phase of SDLC,especially the regression testing and integrated testing phases. The basic assumption is that,defect occur at any stage in the testing phase or failure mode is the best indication of the software reliability.

$$F(t)=(k+1)(\lambda.e^{-\lambda t})$$

Naïve bayes is a very effective classification technique to predict the existence of defects based on the training samples. A naïve Bayes model considers bug prediction as a binary classifier i.e. it trains and predict predictor by analyzing historical metric data.If the attribute types in the metric data are mixed type , then it is difficult to predict the defects due to missing values or uncertain data.

KNN method to judge the defect rate in software status and events. They try to give the software defect rates using some statistic techniques. With the data mining techniques more mature and widely used, for analysis and mining the hidden information in the software development repository become a hot research topic. The usual ways which use data mining techniques in this domain include Association Rules, Classification and Prediction, Clustering. Classification means to build defects prediction model by learning the already existed defects data to predict the defects in future version of software. [8] use this method to improve the efficiency and quality of software development. Some other researches include raised to predict the status and the number of software defects. The current software defect prediction, mainly uses the software metrics to predict the amount and distribution of the software defects. The research method of software defect classification, prediction is based on the program properties of the historical software versions, to build different prediction models and to forecast the defects in the coming versions.

Dynamic analysis techniques can be categorized into three independent layers. First layer is a systematic testing layer. This layer is to execute target programs within policies. These policies aim to reach error states effectively. Second layer is an information extraction layer. The information on the internal behaviors of the target programs is extracted to be used for the program correctness checking. At third layer, the monitors generate an abstract model of the target program from the extracted information and then verify the abstract model to detect possible errors in the program. Dynamic analysis techniques, share the limitations of testing inherently. Dynamic analysis cannot support complete analysis of target programs since it uses monitored partial behavior of the target programs. The other limitation is that dynamic analysis techniques are difficult to be applied unless target programs are complete. Dynamic analysis techniques require executable environments and test cases[7-9].

In [3] importance of different software metrics with prediction model.In this model, they implemented correlations and metric occurrences in the bug prediction model by using different algorithms and the number of bugs in each metric was computed. [4] implemented object oriented metrics to measure the object oriented software quality.It was found that models which are built on coupling and complexity are more precise and accurate than the models build on other metrics.
[5] , designed a model that describes the prediction of 90 releases in open source projects and other projects on academics to perform clustering algorithm. They implemented similarity cluster measures to group the metrics in the design and implemented phases. Statistical tests are used to validate the cluster in each group of metrics.[6] implemented the principal component analysis to reduce the simple multi-collinear complexity to un-correlated measures of orthogonal complexity.

[6] Proposed a model to predict bugs and their levels with high, medium and low severity faults and found that the high severity faults are less accurate as compared to the traditional models at different severities.

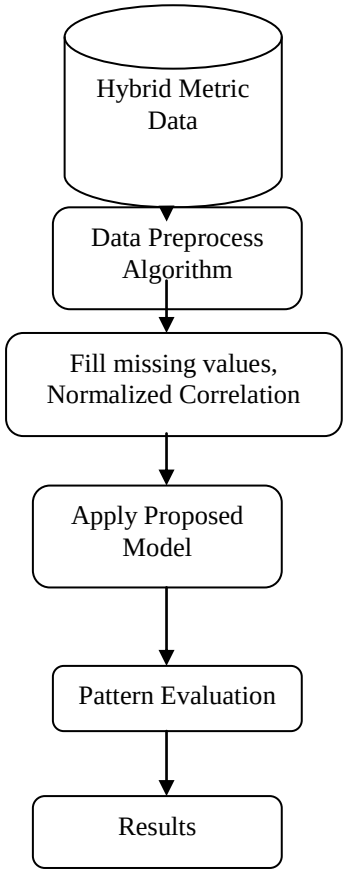
Regression technique is aimed to predict the quantity and density of software defects. Classification technique is aimed to determine whether software module (which can be a package, code, file, or the like) has a higher defect risk or not. Classification usually learns the data in earlier versions of the same project or similar data of other projects to establish a classification model. The model will be used to forecast the software defects in the projects which need to be predicted. By doing this, we can make more reasonable allocation of resources and time, to improve the efficiency and quality of software development[9-12].

Main Objectives of this paper:

- Remove noise in hybrid dataset using correlated based normalization.
- Muli-Variate decision patterns for defects relationship.
- Handle mixed data-type and uncertain decisions.

Proposed Model

In this model, multi-phase metric data was given input to the proposed model for preprocessing. In this framework as shown in Fig 1, input software metric data with a large number of attributes and values are given input to the filtering technique. Filtering algorithm handles missing data and normalized correlation computations for data transformation. After the data transformation, output filtered data is used for the hybrid Bayesian based ranking model to predict and rank the features for pattern mining. Each pattern in the hybrid model is evaluated using F-measure, FP, TP and accuracy of performance evaluation. Finally, decision patterns relevant to set of metrics are evaluated for defect prediction.



Algorithm1: Preprocessing Model

Input: MultiVariated Metrics
Thres: Metric Threshold
Output: Filter Data
Procedure:
Read metrics input data as D.
For each metric M[i] attribute in D
Do
For each instance I[j] in M[i]
Do
If(I[j]==null & M[i+1]!=null)
Then
I[j]=(Mean(M[i])+S.D(M[i]))/(2*Max{M[I],M[I+1]});
End if
If(I[j]==null & M[i+1]==null)
Then
I[j]=(Mean(M[i])+S.D(M[i]))/(2*Max{M[I],M[I-1]});
End if
End for
End for
For each pair of metrics M[i] and M[i+1]
Compute Normalization as
NM[i]= Normalize(M[1]);
NM[i+1]=Normalize(M[i+1)];
NML=addList(NM[i]);
NML=addList(NM[i+1]);
done
done
Sort normalized metrics list NML in ascending order.

For each pair of normalized metrics
Do
Compute Predictive correlation between the two metrics as
Predictive Correlation PC=Corr(NML[i],NML[i+1])/ $\sum_{i=i}^{N}$ Prob(NML[i] / NML[i+1]) ;
If(PC>thres)
Then
D' =addMetric(NML[i],NML[i+1],PC);
End if
Done

Algorithm 1, describes the hybrid preprocessing algorithm on the hybrid metric dataset for noise and data transformation. Algorithm reads the input data and checks the each instance for missing values. If the instance value is missing, then it is replaced with the equation (1) or equation (2). After replacing the missing instances, each pair of metrics is normalized to remove the un-certainty. Afterwards, compute the predictive correlation between two metrics and check the condition with the user defined threshold.

Algorithm-2: Hybrid Bayesian Ranking Based Pattern Miner(HBRBPM)

Input : Filtered Data D'
Output: Decision patterns
Procedure:

Step 1: Choose a pair of metrics with highest correlated features.m1,m2 be the two metrics with the highest predictive correlated measures.

Step 2: Compute rank based attribute measure as follows

Rank(m1)= $\sigma 1 = -(m_1^2 \log(m_2)) * e^{-0.5*PC(m2,m1)}$

Rank(m2)= $\sigma 2 = -(m_2^2 \log(m_1)) * e^{-0.5*PC(m1,m2)}$

$\sigma = \max\{\sigma 1, \sigma 2\}$

Step 3: $if\ (\sigma > \lambda)$
Then
Create a node with $\sigma(\max\{\sigma 1, \sigma 2\})$ as root

 ISSN: 2321-8169
6360 - 6365

Else

Compute the predictive correlation and gain computation between the other metrics.

End if

Step 4: Repeat the steps 2,3 until all metrics
Step 5: Validate the test using F-measure and t-test.
Step 6: Extract rules from the tree.
Step 7: Display results.

Sample Data:

@attribute ACTION numeric
@attribute CONDITIONAL numeric
@attribute CONTINUANCE numeric
@attribute IMPERATIVE numeric
@attribute OPTION {1,2,3}
@attribute RISK_LEVEL numeric
@attribute SOURCE numeric
@attribute WEAK_PHRASE numeric
@attribute DEFECT {Y,N}

@data
1,1,0,0,0,1,1,0,Y
1,0,0,0,0,1,2,0,Y
1,1,0,1,0,1,1,0,Y
2,0,1,1,0,2,1,0,Y
3,1,1,1,0,2,1,0,Y
1,0,1,0,0,2,6,0,Y
1,1,1,1,0,2,3,0,Y
2,0,1,1,0,3,2,1,Y
2,0,0,0,0,1,2,0,Y
1,1,0,1,0,1,1,0,Y
1,1,0,0,0,2,3,0,Y
1,0,0,1,0,3,6,1,Y

Data 2:

2,1,0,15,14,50,1,91,false
5,1,0,20,17,50,5,82,false
4,1,0,3,16,0,4,17,false
16,1,0,8,28,82,14,181,false
3,2,0,17,73,22,2,73,false
1,1,0,11,13,0,1,58,false
18,3,0,9,49,83,18,307,false
4,2,0,4,24,33,0,25,false
3,0,6,0,3,100,0,6,false
6,1,0,5,18,62,5,45,false
3,2,0,5,23,33,0,28,false
5,2,0,5,25,72,0,91,false
5,3,0,5,36,70,5,54,false
4,1,0,1,16,0,3,18,false
4,3,0,5,41,91,3,37,false
5,1,0,5,17,60,3,91,false
3,2,0,5,23,16,0,43,false
2,1,0,15,14,50,1,91,false
5,1,0,20,17,50,5,82,false
4,1,0,3,16,0,4,17,false
16,1,0,8,28,82,14,181,false
3,2,0,17,73,22,2,73,false
1,1,0,11,13,0,1,58,false
18,3,0,9,49,83,18,307,false
4,2,0,4,24,33,0,25,false
3,0,6,0,3,100,0,6,false
6,1,0,5,18,62,5,45,false
3,2,0,5,23,33,0,28,false
5,2,0,5,25,72,0,91,false
5,3,0,5,36,70,5,54,false

Data 3:

```
@RELATION nickle
@ATTRIBUTE filetype {documentation,images,i18n
,ui,multimedia,code,build,devel-doc,unknown}
@ATTRIBUTE filepath string
@ATTRIBUTE revision string
@ATTRIBUTE author_id numeric
@ATTRIBUTE lines_added numeric
@ATTRIBUTE lines_removed numeric
@ATTRIBUTE cvs_removed {0,1}
@ATTRIBUTE cvs_silent {0,1}
@ATTRIBUTE external {0,1}
@ATTRIBUTE commit_date date "yyyy-MM-dd
@DATA
build,.cvsignore,1.5,1,2,1,0,0,0,"2004-05-28 00:36:40"
build,.cvsignore,1.4,1,4,0,0,0,0,"2004-04-01 19:49:56"
build,.cvsignore,1.3,2,4,0,0,0,0,"2003-07-10 17:43:53"
build,.cvsignore,1.2,1,4,1,0,0,0,"2003-06-07 18:21:18"
build,.cvsignore,1.1,2,21,0,0,0,0,"2002-08-07 06:18:45"
devel-doc,AUTHORS,1.2,3,1,1,0,0,0,"2003-05-13 18:18:05"
devel-doc,AUTHORS,1.1,3,2,0,0,0,0,"2001-02-11 01:27:10"
devel-doc,COPYING,1.2,1,1,1,0,0,0,"2004-02-27 03:50:15"
devel-doc,COPYING,1.1,3,29,0,0,0,0,"2001-02-11 01:27:10"
```

Proposed Experimental Results:

lines_removed <= 542.4 -> filetype != documentation
lines_removed < 678.0 -> filetype != documentation
lines_added <= 1011.0 AND lines_removed < 135.6 -> external != 1
lines_added <= 404.4 -> filetype != documentation
filetype != documentation AND lines_added < 606.5999999999999 -> lines_removed < 135.6
lines_added <= 808.8 AND lines_added <= 606.5999999999999 -> external != 1
lines_removed < 678.0 AND lines_removed <= 271.2 -> lines_added <= 1011.0
lines_removed < 135.6 AND lines_removed < 678.0 -> lines_added <= 404.4
lines_removed < 135.6 -> filetype != images
filetype != i18n -> external != 1
lines_added < 1011.0 AND lines_removed <= 135.6 -> external != 1
lines_added <= 404.4 -> filetype != images
lines_added < 1011.0 AND external != 1 AND lines_removed < 271.2 -> filetype != images
lines_added < 1011.0 -> filetype != documentation
lines_added <= 1011.0 AND lines_added < 808.8 -> filetype != images
lines_added < 202.2 AND lines_removed <= 135.6 AND lines_removed < 271.2 -> filetype != images
lines_removed < 271.2 AND lines_removed < 135.6 -> lines_added <= 404.4
filetype != documentation AND lines_added < 606.5999999999999 -> lines_removed <= 135.6
RFC <= 862.0 -> Bug-count != false
NOC <= 38.0 AND NPM <= 214.0 -> CBO <= 125.0
NOC <= 38.0 -> Bug-count != false
RFC >= 0.0 -> NOC <= 38.0
RFC >= 0.0 -> WMC <= 351.0
CBO <= 125.0 AND NPM <= 214.0 -> WMC <= 351.0
LOC <= 5317.0 AND DIT >= 0.0 -> RFC >= 0.0
NOC <= 38.0 -> WMC <= 351.0
NOC <= 38.0 AND DIT >= 0.0 -> RFC <= 862.0
DIT >= 0.0 AND NPM <= 214.0 -> WMC <= 351.0
Bug-count != false -> CBO <= 125.0
WMC <= 351.0 -> RFC >= 0.0
DIT >= 0.0 AND RFC <= 862.0 -> Bug-count != false
LOC <= 5317.0 AND NPM <= 214.0 -> WMC <= 351.0
LOC <= 5317.0 -> CBO <= 125.0
RFC >= 0.0 -> NPM <= 214.0
Bug-count != false -> WMC <= 351.0
NOC <= 38.0 AND RFC <= 862.0 -> NPM <= 214.0
NOC <= 38.0 AND Bug-count != false -> DIT >= 0.0
NOC <= 38.0 AND NPM <= 214.0 AND DIT >= 0.0 -> RFC <= 862.0
RFC <= 862.0 -> CBO <= 125.0
DIT >= 0.0 -> LOC <= 5317.0
NOC <= 38.0 -> LCOM >= 0.0
NPM <= 214.0 AND WMC <= 351.0 -> RFC <= 862.0
WMC <= 351.0 AND LCOM >= 0.0 -> RFC >= 0.0
CBO <= 125.0 AND RFC >= 0.0 -> Bug-count != false
RFC >= 0.0 AND NPM <= 214.0 -> CBO <= 125.0
LCOM >= 0.0 AND NPM <= 214.0 -> Bug-count != false
CBO <= 125.0 AND NPM <= 214.0 AND DIT >= 0.0 -> RFC <= 862.0
WMC <= 351.0 AND RFC >= 0.0 -> Bug-count != false
LOC <= 5317.0 AND DIT >= 0.0 -> RFC <= 862.0
lines_removed <= 135.6 AND lines_removed <= 678.0 -> lines_added <= 404.4
lines_added <= 404.4 AND lines_added <= 606.5999999999999 -> external != 1
lines_removed <= 678.0 AND lines_removed <= 135.6 -> filetype != i18n
lines_removed <= 678.0 AND lines_removed <= 135.6 AND external != 1 AND lines_added <= 202.2 -> filetype != documentation
lines_removed < 135.6 -> external != 1
lines_added <= 202.2 AND external != 1 AND lines_removed < 271.2 -> filetype != images
lines_removed < 542.4 AND lines_removed < 271.2 -> lines_added < 606.5999999999999
lines_added < 1011.0 -> lines_removed < 135.6
lines_removed <= 678.0 AND lines_removed <= 135.6 AND lines_removed < 271.2 -> filetype != images
filetype != documentation -> lines_removed <= 678.0
filetype != images AND filetype != unknown -> lines_added <= 202.2

Performance Measures:

Table 1: Uncertain data preprocessing

Datasize	MissingValues	FilterTime(secs)
#500	12	16
#1000	16	18
#1500	19	21
#2000	25	28
#5000	28	34

Table 1, describes the different data sizes and its missing values and filter time

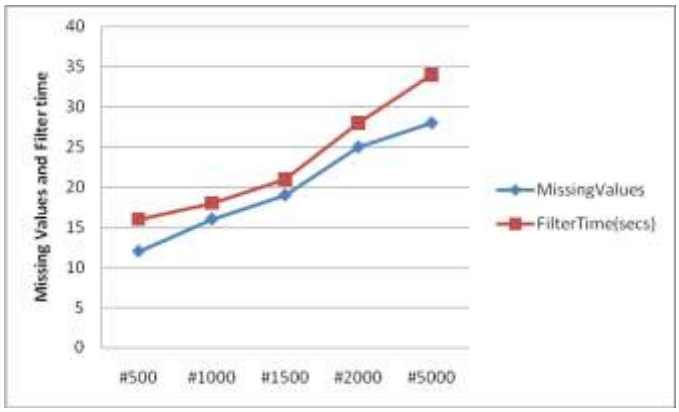


Fig 2: Comparison between datasize Vs Missing values and Filter time

Table 2: Performance analysis of Proposed model with the traditional models.

Datasize	KNN	Predictive Bug detection	Regression Based Bug prediction	HBRBPM
#500	0.87	0.85	0.798	0.96
#1000	0.91	0.867	0.814	0.9524
#1500	0.85	0.827	0.845	0.917
#2000	0.867	0.819	0.842	0.9713
#5000	0.891	0.84	0.86	0.939

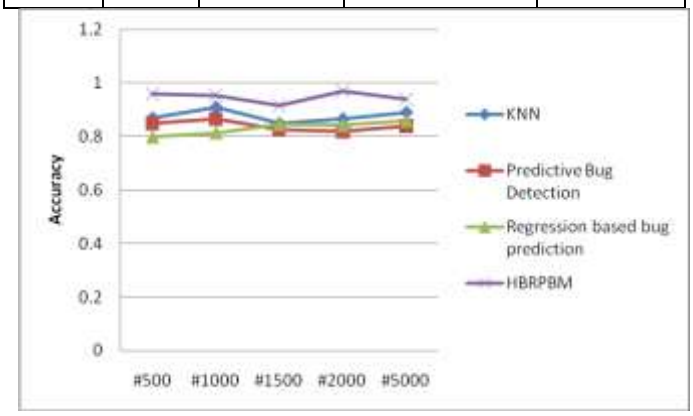


Fig 3: Comparative analysis of proposed model with existing models.

V.CONCLUSION

In this research work, we have used data in different software life cycle phases for defect prediction.In this proposed approach, we have performed robust preprocessing and defects detection algorithm on the metrics data. This approach effectively handles the uncertain data and transform the data for defect detection. Finally, the proposed defect detection model was applied to the transformed data to detect the metric decision patterns. In future , this work can be extended to high dimensional data with more than one project metrics.

REFERENCES

[1] G.Abaeia , A.Selamata , H.Fujitab , "An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction", Knowledge-Based Systems, vol. 74, (2015), pp. 28-39.
[2] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction",Applied Soft Computing, vol. 27, (2015), pp. 504-518.
[3] I. H. Laradji, M.Alshayeb, L.Ghouti, "Software defect prediction using ensemble learning on selected features. Information and Science Technology", vol. 58, (2015), pp. 388-402.
[4] W. Zhang, Y. Yang, Q. Wang, "Using Bayesian Regression and EM algorithm with missing handling for software effort prediction", Information and software technology, vol. 58, (2015), pp. 58-70.
[5] P. He, B. Li, X. Liu, J. Chen, Y. Ma, "An empirical study on software defect prediction with a simplified metric set", vol 59, (2015), pp. 170-190.
[6] V. Ajay Prakash, D. V. Ashoka, V. N. ManjunathAradya, "Application of Data Mining Techniques for Defect Detection and Classification", Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014, Advances in Intelligent Systems and Computing, vol. 327, (2015), pp. 387-395
[7] A. Kaur and I. Kaur, "Empirical Evaluation of Machine Learning Algorithms for Fault Prediction", LectureNotes on Software Engineering, vol. 2, no. 2, (2014).
[8] T. Menzies, J. Greenwald & A. Frank (2007) "Data mining static code attributes to learn defect predictors", IEEE Transaction Software Engineering., Vol. 33, Issue 1, pp. 2-13.
[9] R. Spiewak & K. McRitchie (2008) "Using software quality methods to reduce cost and prevent defects", Journal of Software Engineering and Technology, pp. 23-27.
[10] D. Shiwei (2009) "Defect prevention and detection of DSP-Software", World Academy of Science, Engineering and Technology, Vol. 3, Issue 10, pp. 406-409.
[11] P. Trivedi & S. Pachori (2010) "Modelling and analyzing of software defect prevention using ODC", International Journal of Advanced Computer Science and Applications, Vol. 1, No. 3, pp. 75- 77.
[12] T. R. G. Nair & V. Suma (2010) "The pattern of software defects spanning across size complexity", International Journal of Software Engineering, Vol. 3, Issue 2, pp. 53-70.