International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

Machine Learning based Test Case Prioritization in Object Oriented Testing

Ajmer Singh, Rajesh Kumar Bhatia, Anita Singhrova

Abstract*: Software maintenance is one of the most expensive activities in software life cycle. It costs nearly 70% of the total cost of the software. Either to adopt the new requirement or to correct the functionality, software undergoes maintenance. As a consequent of maintenance activities, software undergoes many reforms. Newly added software components may affect the working of existing components and also may introduce faults in existing components. The regression testing tries to reveal the faults that might have been introduced due to these reformations. Running all the prior existing test cases may not be feasible due to constraints like time, cost and resources. Test case prioritization may help in ordered execution of test cases. Running a faulty or fault prone component early in testing process may help in revealing more faults per unit of time. And hence may reduce the testing time. There have been many different criteria for assigning the priority to test cases. But none of the approaches so far have considered the object oriented design metrics for determining the priority of test cases. Object oriented design metrics have been empirically studied for their impact of software maintainability, reliability, testability and quality but usage of these metrics in test case prioritization is still an open area of research. The research reported in this paper evaluates subset of CK metrics. Metrics considered from CK suite include Coupling between objects (CBO), Depth of Inheritance tree (DIT), weighted methods per class (WMC), Number of children (NOC), and Response for a class (RFC). Study also considers four other metrics namely publically inherited methods (PIM), weighted attributes per class (WAC), number of methods inherited (NMI) and number of methods overridden. A model is built based on these metrics for the prediction of software quality and based on the quality measures software modules are classified with the help of Support Vector Machine (SVM) algorithm. The proposed approach is implemented in WEKA tool and analysed on experimental data extracted from open source software. Proposed work would firstly help the tester in identifying the low quality modules and then prioritize the test cases based on quality centric approach. The work also attempts to automate test case prioritization in object oriented testing. The results obtained are encouraging.

Keywords: Object oriented Software Testing, Machine Learning, Test case Prioritization, Object oriented metrics. Regression testing.

Manuscript published on 30 September 2019
* Correspondence Author
Ajmer Singh*, CSE Department, DCRUST Murthal, Sonipat, India. Email: ajmer.saini@gmail.com
Rajesh Kumar Bhatia, CSE Department, PEC University , Chandigarh, India. Email: rbhatiapatiala@gmail.com
Anita Singhrova CSE Department, DCRUST Murthal, Sonipat, India. Email: nidhianita@gmail.com

I. INTRODUCTION

Object oriented testing differs from the traditional testing as it entails the aspects like inheritance, polymorphism and data abstraction [1]. Also the quality of object oriented software is closely related with its design measures like object oriented inheritance, coupling and cohesion [2]. Many object oriented design metrics have been proposed in the literature. Some of the mostly used metrics include Chidamber & Kemerer (CK) [3], Tang & Chen [4], Henderson [5], Li and Henry [6], Li [7], MOOD [8] and QMOOD [9]. Detailed description of these metrics is given in study [10].These metrics have been used in different contexts like prediction of faults [11], prediction of maintainability[6], and measure of complexity. The Chidamber- Kemerer (CK) metrics are the most used metrics in the literature. Various research studies [4], [12], [13] have investigated CK metrics as software quality indicators. Some object oriented features help in revamping the software quality whereas some may impair it. High quality software should have a low-coupled and highly cohesive design as one of the important parameters. Low coupling and high cohesion are the important software design principles [14]. And consequently a module may impart either positive or negative impact on software quality. Regression testing involves the retesting of software modules to unearth the faults that might have been introduced due to software reforms. Generally testing data of prior version of the software is available with the tester. The information about the coupling and cohesion of prior version of the software may be helpful in identifying and localizing the fault prone components in subsequent version. Also the measure of couplings among the software components in the software can help in ranking the components as per the quality aspect of final software product. A low quality component needs to given priority when testing is to be performed. There are evidences [13], [15], [16] that quality of software is influenced by the object oriented design metrics. Considering class level unit testing, different weights can be assigned to the classes as per their quality measures. Test case prioritization aims to execute the test cases in an order that helps in meeting some predefine testing goal. The goal may be, executing the fault prone components early, so that higher number of faults can be detected early in time. Since metrics like CK metrics, can be statistically estimated from the code of software, machine learning techniques can be employed to the test case prioritization in this context.

This research work proposes a quality centric approach to the test case prioritization for object oriented systems. The proposed work statistically evaluates the previous versions of software under test (SUT) to quantify the quality of individual modules. The quality information derived from object oriented metrics is used to prioritize the test cases. Support vector machine learning technique has been deployed to classify the different modules. The proposed work not only presents machine learning based approach to test case prioritization but also helps in improving the quality of software by identifying low quality modules from it. The paper is organized as follows: Section II reviews related work available literature in the domain. Section III explains proposed methodology of our study. Section IV outlines the empirical evaluation. And lastly, section V concludes the study and suggests future directions.

II. RELATED WORK

Testing of object oriented software is different from the procedural ones. Testing in object oriented software may require different levels of testing to uncover the possible faults. Object oriented testing involves some specific characteristics of the software which are not present in procedural software. Features like Inheritance, polymorphism and encapsulation benefit the software development through code reuse and abstraction. But also some undesirable consequences may be introduced because of these features. Increase coupling in the software is one such possibility. While addressing the problem of test case prioritization, these specific characteristics need to be considered. Not much research work has been reported in the literature related to test case prioritization in object oriented testing. It is observed that most of work on test case prioritization in object oriented testing been carried out by model based approaches. Some of the relevant studies are depicted in following text.

Authors in [17] proposed test case prioritization for system testing. The authors have considered sequence diagrams to generate and prioritize test cases for object oriented systems. Authors converted sequence diagram into sequence graph (SG) to represent the test scenarios. A scenario coverage criterion based on simple paths has been applied to the SG to find weighted paths for every test scenario. Object oriented slicing based prioritization was solicited by [18]. Class slicing technique was utilized by the authors to isolate the interesting parts from rest of program. Test scenario was modeled using Test Dependency Graph (TDG). Test cases are ordered by identifying the strongly connected components (SSCs) in the TDG. The work of [19] and [20] is based on dependence model of object oriented software. A modified dependence model generated for every new version of the software. A mapping of test cases and modified components is derived. The test cases that cover the higher number of affected components in the model are given higher priority. The authors in [21], [22] have suggested that software complexity negatively affects its maintainability. And also there should be low coupling among the software modules. Sometimes classes are indirectly associated and these associations bring into existence indirect coupling among those classes. These complex coupling associations between classes make testing and maintenance difficult and costly [6], [23]. Also there exist some research attempts to estimate the quality of software from different object oriented metrics. The correlation of object oriented indirect coupling and maintenance efforts is studied by [24]. Their study concluded that maintenance effort is directly proportionate to indirect coupling among software modules. The study in [25] considered Martin`s [26] metrics and estimated indirect coupling among the software components. Authors also proposed new metrics for measuring indirect coupling. The research work in [27] evaluated the McCabe and Halstead metrics sets for prediction of fault prone modules from NASA data sets. A total of 21 different metrics derived from the code were examined and as per their investigations, SVM based prediction is more accurate than other counterparts namely random forest and decision tree. A machine learning approach endorsed by Artificial Neural Networks (ANN) was envisaged by [28]. The authors trained ANN to predict fault prone modules from NASA data sets. SVM based approach then classified the modules into fault prone and fault free modules.

Also there are some research studies [12], [29], [30], that attempt to predict the quality of the software by analyzing the software from different perspectives. A SVM based Quality prediction model was formulated by [29]. They studied how Change Reports (CRs) are related with McCabe's cyclomatic complexity and Belady's metric. In their study, CRs were used as indicators of faults in modules. Effect of CK metrics on the software quality was investigated by [30]. In their approach, authors put some thresholds on the CK metrics. Different metric values are then compared against these thresholds and accordingly quality controlling values are determined for different software modules. The study in [12] proposed a model to that indirectly determines quality of the software based upon reliability, testability, maintainability, reusability and efficiency. The model derived these factors from the CK metric suite.

Also some studies [31] and [32] considered more than one metric suites together for studying their integrated impact on measurable software attributes. The authors evaluated CK and Li and Henry metrics on software maintainability. In [31], maintainability was estimated by number of changes in three years of software life whereas study of [32] estimates maintainability through unsupervised learning. Not all the possible faults in the software have equal severity levels. Some faults may be more harmful than others. Thus, when determining the priority of test cases, severity of faults should also be taken into consideration. Study in [33] empirically deliberated the consequences of CK metrics in fault proneness of a module. Faults with different severity levels were assumed to be present in software. Authors applied SVM to segregate the modules in fault prone and fault free classes. Also the studies [34], and [35] appraise object oriented metrics through machine learning and statistical techniques quality prediction of the software. Authors in [34], developed a prediction model based on CK and QMOOD metrics for data sets extracted from an open source software. Out of the six machine learning methods, random forest and bagging methods performed better as per their study.

In another study [36], six sequential versions of Ant software were analyzed to establish relationship between object oriented metrics and the quality of the software. Authors employed a linear regression model to associate CK metrics with the quality. Analysis of prior researches substantiates the beliefs that object oriented metrics considerably control the quality of that software and also may be used for assessing the software for reliability and maintainability.

III. PROPOSED METHODOLOGY

It is evident from the analysis of literature, that object oriented metrics are closely associated with software reliability, maintainability and quality. The research work proposed here, asses the quality of the constituent classes of the software on the basis of object oriented metrics. On the basis of measured quality, classes are categorized into good quality and poor quality categories. The figure 1 shown below gives brief idea of the proposed methodology. Where, first step involves extraction of the values for various object oriented metrics of the study. The second step determines the quality of various software modules. Data extraction and quality determination was performed by JMT[1] tool. JMT provides the metric data for project level, class level and method level granularity. A class level granularity level was considered for this study. The third step involves analysis of relationships between the metrics of the study and the predicted quality. A regression coefficient based analysis was performed for this. In next step, different software modules were classified using SVM technique. The details of classification step are given in next section. The modules are segregated into two classes after the classification step. The boundary that isolates the two classes is obtained. It comprises of support vectors. In the next step, Euclidian distance between each data point and the support vector is calculated. Next step involves sorting the modules as per the distance from the support vectors. Finally the modules are ranked as per their quality aspects. The steps are further elaborated in next section.

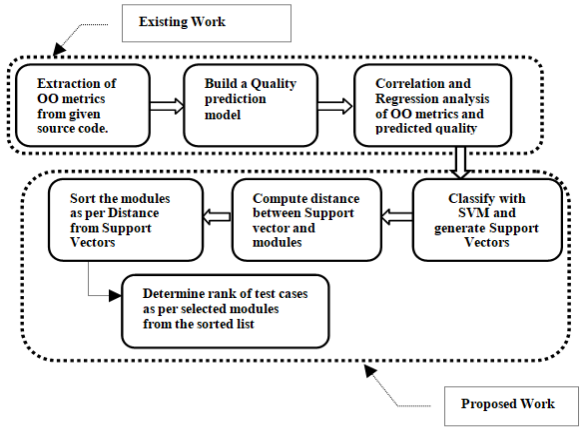

Fig 1 . Proposed Methodology

A. Proposed algorithm for sorting of modules

The fig. 2 shown below depicts the steps of proposed algorithm for sorting of modules. The quality vector Q_m is function of different object oriented metrics for a module.

Proposed Algorithm

1	*for every module m_i get the quality vector* $Q_m = <X_1, X2, X3,..., X_n>$
2	*Get the score (Θ)of the Q_m by applying scoring function classify the Q_m*
3	*{ if $\Theta<0 \Longrightarrow$ category = -1 else category =+1 }*
4	*For every module m_i get the Euclidian distance d_i of Q_m from hyper plane and store d_i as + d_i if class is +1 else store the d_i as ($-d_i$) into a list (L). Also save the module index (i) in L. /* treating -ve values as indicators of low quality */*
5	*Sort the L in increasing order on the distance values.*

Fig. 2. Proposed algorithm for sorting of modules

The variables $X_1, X2, X3... X_n$ represent the metrics values for any module m. The scoring function in second step is needed to classify the current data point. The value (Θ) of score function determines the class of data point. It should be noted that, depending upon the value of (Θ), module may belong to either a (+ve) category or a (-ve) category. Where (+ve) category represents the good quality and (-ve) represents the poor quality. Consequently, larger the distance of module in (+ve) direction, higher would be quality of the module and larger the distance in (-ve) direction, lower would be quality of the module. In other words, the magnitude of distance and direction of sign gives the quantitative measure of quality. Rest of the steps are self explanatory. The process to get support vectors and to compute distances has been explained in experimentation part of this work.

IV. EMPIRICAL EVALUATION

The experimental work including building the prediction model, training and validation and was performed on WEKA [37] tool. WEKA is open source software. A set, comprising 60 classes from XML Security 1.4.0, was used as training data. The data related to version 1.4.1 was used as experiment data.

A. Data Collection

This study uses the data sets extracted from Open Source software "XML Security". XML-security is used for implementing XML signature and encryption standards. It is implemented in JAVA and its size is 16800 lines of code. We collected the data sets for three versions of XML Security viz. xmlsec-1.4.0., xmlsec-1.4.1 and xmlsec-1.4.2. It was obtained from Software-artifact Infrastructure Repository [38]. The three versions of software consist of 317, 321 and 341 classes respectively. The three versions of the software were obtained in .jar format.

B. Independent Variables

From the literature review it was found that object oriented metrics namely Publically Inherited Methods (PIM), Weighted methods per class (WMC), Weighted attributes per class (WAC), Number of methods inherited (NMI), Number of Methods overridden (NMO) Depth of Inheritance (DIT), Number of Children (NOC), Coupling Between Objects (CBO) and Response for a Class (RFC) significantly affect the quality software.

In order to study the influence of these metrics on quality the study considered them as independent variables. The data sets these independent variables, was extracted with the help of JMT [39] tool. JMT can generate the data at package level, class level and method level of the software. JMT also produced the 'name', 'type' and 'package' information of the each Java class analyzed. As these attributes do not influence the analysis of independent and dependent variables, so these attributes were removed from data sets in data pre-processing step. Also, all missing values were set to 0 in the pre-processing step. Data was normalized to avoid the problem of over fitting.

C. *Dependent Variable*

The predicted quality of individual module is variable of interest for this study. JMT tool, estimates the quality of software at class, method and package level. Software quality of different classes in XML security software is taken as dependent variable. Quality predicted by JMT is given as percentage ranging from 0 to 100. The values so obtained were normalized on a scale 0 to 1.

D. *Statistical description*

Statistical analysis of the two different versions of the SUT is necessary in order to know the inter dependence of these versions. The statistical information about the classes of the subject software is shown in table I, II and III below. All the values are the numbers. The data from tables surmise that various statistical measures like minimum, maximum, mean and standard deviation do not vary much in inter version data. So the analytical inferences of one version may be applicable to another subsequent version as well. It also substantiate the fact, that test cases ranked on the basis of one version of software may be applied to applied to another without affecting the testing much. A regression model for each data set is represented by equations (1) to (3).

Table -I: Statistical Analysis of XML-Security 1.4.1

Metric	Min	Max	Mean	Standard Deviation
PIM	0	73	7.925	10.143
WMC	0	65	7.433	7.914
WAC	0	65	2.941	6.038
NMI	0	43	5.953	9.051
NMO	0	23	1.109	2.484
DIT	0	5	2.368	1.342
NOC	0	25	0.461	2.175
CBO	0	52	10.062	10.193
RFC	0	164	22.310	26.343
Quality	0.48	1.00	0.882	0.114

Regression model for XML-Security 1.4.1 is as following

Quality = -0.0035PIM - 0.0022 WMC-0.0027WAC - 0.0018NMI -0.0065NMO + 0.0064DIT -0.006 NOC - 0.0049CBO -0.0002RFC + 0.9936 (1)

Table -II: Statistical Analysis of XML-Security 1.4.0

Metric	Min	Max	Mean	Standard Deviation
PIM	0	73	8.019	8.019
WMC	0	65	7.495	7.495
WAC	0	65	2.975	2.975
NMI	0	43	6.028	6.028
NMO	0	23	1.091	1.091
DIT	0	5	2.372	2.372
NOC	0	25	0.465	0.465
CBO	0	52	10.136	10.136
RFC	0	164	22.316	22.316
Quality	0.48	1	0.88	0.88

Regression model for XML-Security 1.4.0 is as following

Quality = -0.0036 PIM -0.0021 WMC-0.0027 WAC -0.0018 NMI -0.0065 NMO + 0.0063 DIT -0.006 NOC -0.0049 CBO -0.0002 RFC + 0.9933 (2)

Table -III: Statistical Analysis of XML-Security 1.4.2

Metric	Min	Max	Mean	Standard Deviation
PIM	0	72	7.657	9.708
WMC	0	65	7.275	7.836
WAC	0	65	2.935	5.964
NMI	0	43	6.176	9.07
NMO	0	23	1.023	2.388
DIT	0	5	2.428	1.356
NOC	0	25	0.481	2.182
CBO	0	56	9.789	10.346
RFC	0	171	22.144	27.585
Quality	0.50	1	0.885	0.114

Regression model for XML-Security 1.4.2 is as following

Quality = -0.0038 PIM -0.0015 WMC -0.0028 WAC -0.0017NMI-0.0066 NMO + 0.007 DIT -0.0062 NOC -0.0047 CBO -0.0004RFC + 0.9917 (3)

E. *Correlation analysis of metrics*

Out of all these independent variables, some variables may be interrelated. Knowledge of such dependencies is essential for identifying the most influencing variables. Also it helps in reducing the high-dimensionality of the data set. A correlation analysis was performed the help of Principal Component Analysis (PCA).

All the metrics values were normalized on scale 0 to 1 before applying PCA. A ten cross validation technique accompanied with attribute ranking search method was adopted as PCA configurations. Also principal components were configured to retain 95% originality of data sets. And it was implemented in WEKA.

The results shown in Table IV below highlight that CBO and RFC are the highly correlated metrics with positive correlation coefficient 0.95. Whereas, metrics WAC and NMI are least correlated with negative regression coefficient -0.20.

Table -IV: Correlation matrix for the metrics

	PIM	WMC	WAC	NMI	NMO	DIT	NOC	CBO	RFC
PIM	**1**	**0.61**	**0.08**	0.23	**0.03**	**0.19**	**0.32**	**0.35**	0.43
WMC	0.61	**1**	**0.31**	**-0.19**	**0.23**	**0.08**	**0.27**	**0.73**	**0.85**
WAC	0.08	**0.31**	**1**	-0.2	**-0.07**	**-0.08**	**0.06**	**0.3**	**0.35**
NMI	0.23	**-0.19**	-0.2	**1**	**0.4**	**0.73**	**0.21**	**-0.15**	**-0.12**
NMO	0.03	**0.23**	**-0.07**	**0.4**	**1**	**0.43**	**0.32**	**0.36**	**0.32**
DIT	0.19	**0.08**	**-0.08**	**0.73**	**0.43**	**1**	**0.16**	**0.23**	**0.2**
NOC	0.32	**0.27**	**0.06**	**0.21**	**0.32**	**0.16**	**1**	**0.28**	**0.31**
CBO	0.35	**0.73**	**0.3**	**-0.15**	**0.36**	**0.23**	**0.28**	**1**	**0.95**
RFC	**0.43**	**0.85**	**0.35**	**-0.12**	**0.32**	**0.2**	**0.31**	**0.95**	**1**

F. *Dimensionality reduction through PCA*

High dimensionality of the data sets hinders analysis of association between the independent attributes and classifying attribute. The PCA detects different patterns in the data sets and the correlations that exist among the variables. Strongly correlated components may be used to reduce the dimensionality of the data set. PCA finds the directions of maximum variance and transforms it to a smaller directional sub-space. The smaller sub-space can replace the original high-dimensional space while retaining all important information of the data sets.

PCA computes mean vector μ for the d -dimensional data set and produces a $d \times d$ covariance matrix. Subsequently eigenvectors and Eigen value are computed. Eigen value and vectors are sorted as per decreasing order of Eigen value. PCA chooses top k vectors know as principal components.

We applied PCA with ten cross validations and Ranker algorithms for ranking the components. To retain the significant information with data set, PCA with threshold of -∞ and variance of 98% was adopted. The two components C_1 and C_2 were obtained. Where C_1 covers 71% variance and C_2 covers 37% variance. The equation (4) and (5) show the component C_1 and C_2 respectively.

$$C_1 = -0.472RFC - 0.442CBO - 0.441WMC - 0.376PIM - 0.29DIT - 0.235NMO - 0.231WAC - 0.227NMI - 0.041NOC \quad (4)$$

$$C_2 = -0.64NMI - 0.459DIT + 0.363WAC - 0.261PIM + 0.252RFC + 0.227CBO + 0.226WMC - 0.114NMO + 0.01NOC \quad (5)$$

G. *Classification by SVM*

SVM is a machine learning algorithm that supports supervised learning [40]. It can be applied for classification as well as regression. In classification, SVM works as a discriminative classifier means it segregates the binary class data into two classes. SVM learns from a given inputs called the 'training data' which are classified with the expected output [41]. Unlike other classifiers, SVM tries to find the best separating line for the two given data sets. The SVM looks for the nearest data points from the two classes, which are called as "support vectors". We applied binary SVM classifier to bisect the data into two classes. Working of SVM is explained as follows

If $\boldsymbol{X}$ be n-dimensional feature vector. Such that $\boldsymbol{X} \in \mathbb{R}^n$

And scalar y be the class such that $y \in \{1, -1\}$, then binary SVM tries to find orthogonal vector $\boldsymbol{W}$, and scalar b such that either

$$\boldsymbol{W}.\boldsymbol{X} + b = 1 \quad (6)$$

or

$$\boldsymbol{W}.\boldsymbol{X} + b = -1 \quad (7)$$

Where $\boldsymbol{W}$ represents the normal vector of hyper plane. SVM tries to maximize the distance between two decision boundaries created by (6) and (7). Geometrically, the distance between the two planes represented by (6) and (7) is $\mathbf{2}\|\mathbf{W}\|$. A binary SVM segregates the data set into two classes such that these classes are separated by maximum distance [42]. The data points nearest to hyper plane constitute the support vectors of two segregations.

In this study, Quality is the classifying attribute of the data sets. After the normalization quality attribute consisted of values ranging from 0 to 1, where 0 is poorest and 1 is best quality of the module. The quality attribute (Q) was standardized to 1 and -1 as per following inferences. For any data point $\boldsymbol{X}$ its quality estimate is denoted by $Q(\boldsymbol{X})$.

$$If\ Q(X) < \mu \Rightarrow Q(X) = -1 \quad (8)$$

$$If\ Q(X) \geq = \mu \Rightarrow Q(X) = 1 \quad (9)$$

Where, $Q(\boldsymbol{X})$ is the quality indicator for any data point $\boldsymbol{X}$ and μ being the overall mean quality of data set. The numeric to nominal conversion was applied to quality attribute to make it Binary-SVM compatible.

SVM with poly kernel classifier and logistic calibrator was applied on the normalized data set.

Quadratic function based kernel is popular form of poly kernel classifiers. This study incorporated second order function for kernel.

After application of SVM, a total of 16 support vectors were generated were obtained for this data set with coefficient (b) =1.7827. Table V below presents the top 11 selected support vectors. Table VI shows gives the accuracy details of the generated support vectors. It can be seen that ROC are around 97% is achieved.

Table - V: Generated Support vectors

Support Vector	CLASS	PIM	WMC	WAC	NMI	NMO	DIT	NOC	CBO	RFC
S1	-1	0.041	0.250	0.076	0	0.071	0.6	0.000	0.596	0.416
S2	+1	0.208	0.510	0.015	0.037	0.071	0.6	0.210	0.259	0.291
S3	-1	0.375	0.297	0	0.444	0.285	0.8	0.000	0.346	0.333
S4	+1	0.291	0.333	0	0	0	0.4	0.000	0.192	0.275
S5	+1	0.208	0.297	0.030	0.037	0.071	0.6	0.000	0.326	0.291
S6	+1	0.333	0.125	0.000	0.259	0.142	0.8	0.000	0.173	0.108
S7	+1	0.291	0.251	0.000	0.259	0.142	0.8	0.000	0.173	0.133
S8	-1	0.166	0.333	0.046	0.037	0.071	0.6	0.000	0.442	0.341
S9	-1	0.291	0.417	0.030	0.037	0.071	0.6	0.000	0.480	0.441
S10	-1	0.208	0.375	0.030	0.259	0.428	0.8	0.000	0.365	0.291
S11	+1	0.416	0.453	0.015	0	0	0.4	0.200	0.211	0.183

Table -VI: Accuracy measure of generated support vectors

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
1.000	0.056	0.917	1.000	0.957	0.930	0.972	0.917	-1
0.944	0.001	1.000	0.944	0.971	0.930	0.972	0.971	+1

Where +1, and -1 are the two categories of the segregations for the software classes. Table above shows the possible support vectors for both the categories; +ve and +ve quality of the classes. The values corresponding to a particular metric signify the weight of coefficient of that metric in support vector equation. For instance the first support vector corresponding to first row of the table can be expanded in the form given in equation (10)

$$S1 = 0.041(PIM) + 0.25(WMC) + 0.0769(WAC) + 0(NMI) + 0.071(NMO) + 0.6(DIT) + 0(NOC) + 0.596(CBO) + 0.416(RFC) \quad (10)$$

All other remaining 10 support vectors can be expanded in same way.

Supports vectors presented in table 5 are of high dimensions and are reduced to three dimensional spaces by applying PCA equations obtained in (4) and (5) respectively. The figure 3 shown below depicts the hyper plane for the poly kernel function after the support vectors were reduced to three dimensional spaces. The hyper plane intersects on z axis in such a way that data points to its left represent -1 class and to the right represents the +1 class.

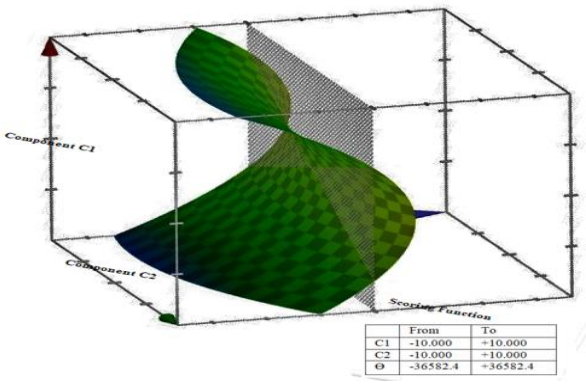

Fig. 3. Hyper plane of Support vectors

H. *Scoring Function*

In order to predict the class of data points, score for each is calculated on the basis of scoring function. The support vectors and kernel function collectively derive the scoring function (ψ) for the data set.

$$\psi = \sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b \qquad (11)$$

Where m is number of data points, y_i is the class label for i_{th} instance and K be kernel function of support vector x. K evaluated on all support vectors obtained. α_i is a coefficient associated with i_{th} instance. And b is scalar value produced by support vectors. In fig. 3, the component C_1 and C_2 are obtained from PCA algorithm and value of ψ is produced by equation (8). The graph is generated with the help of google graph-plotter [43].

I. *Prioritizing the test cases as per quality metrics*

Proposed work was evaluated on a set of six test cases as shown in Table VII. A pair of support vectors: one from each category was selected. Euclidean distance between the class covered and its corresponding support vector was determined. Results indicate that distances from the corresponding support vector and quality measures are highly related. Testing can be performed on these classes on the on the basis of distance from the hyper plane.

Table – VII: Evaluation results

Test case Id	(C) Classes covered	Quality measure Q(C)	Category of module	Distance from S1	Distance from S2
T1	DOMXML SignatureFactory	0.71	(-1)	1.097	NA
T2	DOMXML Signature	0.62	(-1)	1.106	NA
T3	ElementProxy	0.52	(-1)	1.175	NA
T4	DSAKeyValue Resolver	0.93	(+1)	NA	0.588
T5	DigesterOutput Stream	0.97	(+1)	NA	0.709
T6	DOMSignature Method	0.86	(+1)	NA	0.561

J. *Quality Improvement Potential (QIP)*

In a software system when the susceptibility of the code to the maintenance problems or proneness to the faults can be estimated in design phase, then the components of the software can be weighted accordingly. If the proneness of component can be compared with some threshold value, the components can ranked as well. When ordering the components for testing, the distance from the threshold value can be useful. The components which are below the threshold may need more attention of the testers and have more potential for improving the quality.

Let us consider a software system with T1, T2, T3...Tn as the set of test cases to be executed and C1, C2, C3, ... Cn be the components code that are to be tested. Further, let Q1, Q2, Q3 ...Q_n be the quality measures for these components. If Θ be the threshold of quality measure for these components. Then a test case, which covers a low quality component, has more potential to improve the overall quality of the software. Moreover for the components which have quality measures less than threshold Θ, their relative differences from Θ can be used to determine the corresponding weights. In the table given below test cases T1, T2,...,T7 with their corresponding class covered is shown. Let the quality threshold (Θ) be of 50 units. Also let us consider that each test case takes single unit of time execution time and it is required that test cases are to be prioritized for execution with in time constraint of 5 units. The Table VIII below shows the example for illustration. And Table IX the performance of the proposed methodology is compared with the random prioritization technique. The metric considered for the performance evaluation is the rate of quality improved per test case executed.

Table –VIII: An illustration

Test case	Class covered	Quality (Q)	Distance from Θ
T1	C1	43	07
T2	C2	35	15
T3	C3	34	16
T4	C4	32	18
T5	C5	34	16
T6	C6	20	30
T7	C7	30	20

Table – IX: Comparison with Random Approach

Random approach	As per proposed approach	% improvement
Random order {T1,T2,T3,T4,T5} and rate of improvement of quality (ρ) = (07+15+16+18+16)/5= 14.4	Selected order {T6,T7,T4,T5,T3} (ρ') = (0+20+18+16+16)/5= 20	$\frac{(\rho' - \rho)\ X\ 100}{\rho}$ $\frac{(20.0 - 14.4)\ X\ 100}{14.4}$ = 38.88%

V. CONCLUSIONS AND FUTURE SCOPE

This research work presents a machine learning based approach to test case prioritization. The study closely examined the correlation of software quality and object oriented metrics. Experimental work was performed by considering class level testing. SVM was employed to segregate the testing classes into preferred and not preferred classes. The study also strengthens the confidence that object oriented metrics control the quality of software significantly. The results not only attract the applicability of machine learning to software testing but also pave way for the automation of software test case prioritization in regression testing.

In this work most of the metrics are related to CK metric and are related to design metrics. The study can be extended for other metrics like project and product based metrics

REFERENCES

1. J. D. McGregor and D. A. Sykes, *A practical guide to testing object-oriented software*. 2001.
2. L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *J. Syst. Softw.*, vol. 51, no. 3, pp. 245–273, 2000.
3. S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

4. M.-H. T. M.-H. Tang, M.-H. K. M.-H. Kao, M.-H. C. M.-H. Chen, and R. Martin, "An empirical study on object-oriented metrics," *Proc. 6th Int. Softw. Metrics Symp.*, 1999.
5. Henderson and Sellers, "Object-Oriented Metrics, measures of Complexity," in *Prentice Hall*, 1996.
6. W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *J. Syst. Softw.*, 1993.
7. W. Li, "Another metric suite for object-oriented programming," *J. Syst. Softw.*, vol. 44, no. 2, pp. 155–162, 1998.
8. F. B. Abreu and R. Carapuça, "Object-Oriented Software Engineering : Measuring and Controlling the Development Process," *4th. Int. Conf. Softw. Qual.*, 1994.
9. J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, 2002.
10. A. Singh, R. Bhatia, A. Sighrova, A. Singhrova, and A. Sighrova, "Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics," in *Procedia Computer Science*, 2018, vol. 132, pp. 993–1001.
11. T. Gyimothy, R. Ferenc, I. Siket, T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, 2005.
12. S. Srivastava and R. Kumar, "Indirect method to measure software quality using CK-OO suite," in *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*, 2013.
13. M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *J. Syst. Softw.*, vol. 76, no. 2, pp. 147–156, 2005.
14. S. Husein and A. Oxley, "A coupling and cohesion metrics suite for object-oriented software," *ICCTD 2009 - 2009 Int. Conf. Comput. Technol. Dev.*, vol. 1, pp. 421–425, 2009.
15. F. Brito e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," *Proc. 3rd Int. Softw. Metrics Symp.*, no. March, pp. 90–99, 1996.
16. R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Softw. Eng.*, 2003.
17. D. D. . Kundu, M. . M. Sarma, D. . D. Samanta, and R. R. . Mall, "System testing for object-oriented systems with test case prioritization," *Softw. Test. Verif. Reliab.*, 2009.
18. J. Jaroenpiboonkit and T. Suwannasart, "Finding a test order using object-oriented slicing technique," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, pp. 49–56, 2007.
19. C. R. Panigrahi and R. Mall, "A heuristic-based regression test case prioritization approach for object-oriented programs," *Innov. Syst. Softw. Eng.*, vol. 10, no. 3, pp. 155–163, 2014.
20. C. R. Panigrahi and R. Mall, "An approach to prioritize the regression test cases of object-oriented programs," *CSI Trans. ICT*, 2013.
21. M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *Proceedings of the Australian Software Engineering Conference, ASWEC*, 2007.
22. M. Perepletchikov and C. Ryan, "A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software," *IEEE Trans. Softw. Eng.*, 2011.
23. S. Almugrin and A. Melton, "Indirect Package Coupling Based on Responsibility in an Agile, Object-Oriented Environment," in *Proceedings - 2nd International Conference on Trustworthy Systems and Their Applications, TSA 2015*, 2015.
24. N. K. Gupta and M. K. Rohil, "Object oriented software maintenance in presence of indirect coupling," *Commun. Comput. Inf. Sci.*, vol. 306 CCIS, pp. 442–451, 2012.
25. S. Almugrin, W. Albattah, and A. Melton, "Using indirect coupling metrics to predict package maintainability and testability," *J. Syst. Softw.*, 2016.
26. R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. 2003.
27. K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, 2008.
28. I. Gondra, "Applying machine learning to software fault-proneness prediction," *J. Syst. Softw.*, 2008.
29. M. R. Lyu, "A Novel Method for Early Software Quality Prediction Based on Support Vector Machine," *16th IEEE Int. Symp. Softw. Reliab. Eng.*, 2005.
30. P. Antony, "Predicting Reliability of Software Using Thresholds of CK Metrics.," *Int. J. Adv. Netw. ...*, 2013.
31. Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *J. Syst. Softw.*, 2007.
32. C. Jin and J. A. Liu, "Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics," in *2010 International Conference on MultiMedia and Information Technology, MMIT 2010*, 2010.
33. R. Malhotra, A. Kaur, and Y. Singh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines," in *International Journal of Systems Assurance Engineering and Management*, 2010.
34. R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *J. Inf. Process. Syst.*, 2012.
35. R. Malhotra and Y. Singh, "On the Applicability of Machine Learning Techniques for Object Oriented Software Fault Prediction," *Softw. Eng. an Int. J.*, 2011.
36. A. Singh, R. K. Bhatia, and A. Singhrova, "Object Oriented Coupling based Test Case Prioritization," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 9, pp. 747–754, Sep. 2018.
37. M. Hall *et al.*, "The WEKA Data Mining Software : An Update," *ACM SIGKDD Explor. Newsl.*, 2009.
38. H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," in *Empirical Software Engineering*, 2005.
39. "JMT: A tool for measuring and judging Java code." [Online]. Available: http://jmt.tigris.org/. [Accessed: 12-Jun-2017].
40. Chih-Wei Hsu, Chih-Chung Chang, C.-J. Lin, and C.-J. L. Chih-Wei Hsu, Chih-Chung Chang, Chih-Wei Hsu, Chih-Chung Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification," *BJU Int.*, 2008.
41. S. Johnson, "Effective feature set construction for SVM-based hot method prediction and optimisation Valli Shanmugam," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 3, pp. 192–205, 2011.
42. H. Yu and S. Kim, "SVM tutorial-classification, regression and ranking," in *Handbook of Natural Computing*, 2012.
43. "graph-plotter." [Online]. Available: https://experiments.withgoogle.com/graph-plotter.

AUTHORS PROFILE

Mr. Ajmer Singh obtained the B.Tech. and M.Tech.degree from Kurukshetra University, Kurukshetra, India in 2004 and 2007 respectively and is pursuing Ph.D. from the Deenbandhu Chhoturam University of Science and Technology (DCRUST), Murthal

He is presently working as an Assistant Professor in the Department of Computer Science and Engineering at DCRUST, Murthal, Sonepat, India with more than 11 years' experience of academic and administrative affairs. He has published more than 20 research papers in various International / National Journals and Conferences of repute as author/co-author.

Dr. Rajesh Kumar Bhatia is a senior member of CSI, and he is also senior member of ACEEE, He is working as professor in Computer Science and Engineering Department, PEC University, Chandigarh. His main research work focuses on Software Testing, Software Engineering and Software Clones Detection. He has more than 20 years of teaching experience and 12 years of Research Experience

Dr. Anita Singhrova is professor, Dean Faculty of Information Technology and Computer Science at Deenbandhu Chottu Ram University of Science and Technology, Murthal, Sonepat, India. She holds a Ph.D degree from GGS Indraprastha University, Delhi, India. She has completed M.E (Computer Science & Engg.) from Punjab Engineering College, Chandigarh, India and B.Tech (Computer Science) from T.I.T&S, Bhiwani, India. She has also been certified as Java Programmer by Sun Microsystems. She possesses around twenty years of teaching experience. Her research interests include network security, mobile computing and heterogeneous networks. She has contributed in various research papers and articles published in various national and international journals and conferences.