Finding error-prone classes at design time using class based Object-Oriented metrics threshold through statistical method

DIPTI KUMARI [1], DR. KUMAR RAJNISH [2]

[1]Department of Computer Science, B.I.T. Mesra, Ranchi , Jharkhand 835215
[2]Department of Information Technology, B.I.T. Mesra, Ranchi , Jharkhand 835215
[1]kumari_dipti0511@yahoo.co.in, [2]krajnish@bitmesra.ac.in

Abstract. A study that how error severity categories depend on the class level software metrics is presented through statistical method. The main purpose of the study is to classify error categories based on the different number of error occurrences in all the three version of Eclipse Project. The study used the all error type to find the software metrics threshold for the three releases of Eclipse project using Receiver Operating Characteristic curves. These thresholds are responsible for making difference between error-free or error prone classes . But, not all the choosen metrics are able to do that, though some of them are capable for that. In future it is not necessary that these software metric thresholds can predict the class will definitely have errors. This approach only provide a scientific way for software engineers to judge designed class is error prone or error free during design time.

Keywords: Object-Oriented metrics, object-oriented design, specificity, sensitivity, thresholds, ROC curve, AUC, Confusion Matrix.

(Received June 11th, 2013 / Accepted August 1st, 2013)

1 Introduction

Different measurements are important tools for achieving quality management in the software development process. Two major measurement types are product metrics which are used to control the quality of the software product (e.g. Defect rates) or process metrics which are used to measure the status and progress of the system design process and to predict future effects or problem areas(e.g. maintenance costs).A common problem in large and complex software systems in that they have errors [2].Preventing errors from being introduced into software systems proves to be a difficult , if not an impossible, task. If we cannot completely prevent errors, we want to know where in a design errors are likely to occur. To achieve this goal(of which classes are likely to have errors in a design),many researchers have studied software metrics are suggested metrics models [3-7].Although some of the metric models proved to be effective in empirical studies, they are difficult to use in practice: it is impractical for software engineers to build and run some metrics models on a daily basis. Much recent research work has empirically investigated the relationship between object-oriented (OO) measures and class fault proneness[8-18][23-27]. Once validated such measure can serve as leading indicators of fault-prone classes. Fault-prone classes can then be targeted for specific quality management action, such as more intensive inspections and testing, or they may even be redesigned.

An appealing operational approach for quality management using OO measures is to develop thresholds. Thresholds are defined as ”heuristic values used to set ranges of desirable and undesirable metrics values for measured software. These thresholds are used to identify anomalies, which may or may not be an actual problem”. For example, we can say that a certain coupling measure has a threshold of seven. If the measured value for a particular class is larger than seven, then we could flag that class as high risk. Kecia A. M. Ferreira et.al. [1] presents results of a study on the structure of a large collection of open-source programs developed in Java, of varying sizes and from different application domains. The aim of their work is the definition of thresholds for a set of OO software metrics, namely: LCOM, DIT, coupling factor, afferent couplings, number of public methods, and number of public fields. They carried out an experiment to evaluate the practical use of the proposed thresholds. The results of this evaluation indicate that the proposed thresholds can support the identification of classes which violate design principles, as well as the identification of well-designed classes. The method used in this study to derive software metrics thresholds can be applied to other software metrics in order to find their reference values. Raed Shatnawi et al[19] validated the OO metrics as measure of design complexity. He also conducted few studies to formulate the guidelines, represented as threshold values, to interpret the complexity of the software design using metrics.

In this paper, they use a statistical model, derived from the logistic regression, to identify threshold values for the Chidamber and Kemerer (CK) metrics. The methodology is validated empirically on a large open-source system—the Eclipse project. The empirical results indicate that the CK metrics have threshold effects at various risk levels. They have validated the use of these thresholds on the next release of the Eclipse project—Version 2.1—using decision trees. In addition, the selected threshold values were more accurate than those were selected based on either intuitive perspectives or on data distribution parameters.. These findings suggest that there is a relationship between risk levels and OO metrics and that risk levels can be used to identify threshold effects. Again, he used the three releases of the Eclipse project and found threshold values for some OO metrics that separated no-error classes from classes that had high-impact errors. Although these thresholds cannot predict whether a class will definitely have errors in the future, they can provide a more scientific method to assess class error proneness and can be used by engineers easily [20]. Raed Shatnawi et al. [21] also showed that power law behavior has an effect on the interpretation and usage of software metrics and in CK metrics [34]. Many metrics have shown a power law behavior. Threshold values are derived from the properties of the power law distribution when applied to open-source systems. The properties of a power law distribution can be effective in improving the fault-proneness models by setting reasonable threshold values[21]. Sarabjit Kaur et al.[22] used logistic regression to investigate the threshold values against the bad smell for the Chidamber and Kemerer [34]metrics at five different levels. Two versions of jfreechart were used as a dataset to validate the study. Only the significantly associated metrics were considered for finding the threshold values. The results indicate that the CK metrics [34] have threshold effects at various risk levels and some metrics have useful threshold value at different levels to identify the bad smell.

We believe that meaningful and useful threshold values for software metrics must clearly and explicitly associated with design factors of interest. For example, if we are interested in reducing the probability of errors in a module, the threshold for module metrics must be associated with module error probability. In this study we empirically identify the dependency of software metric threshold value on class error probability using Receiver Operating Characteristics (ROC). However, this method has been used to make decisions about diagnostics in radiology to distinguish between healthy and ill subjects [23], and in clinical medicine [24].Here, classes are assumed as patient and different software metrics threshold values are the test which will indicate the illness of classes(means a class having software metrics value greater than threshold value is more error-prone compared to those classes having software metrics value less than threshold value).

The rest of the paper is organized as follows: Section 2 deals with the Experimental design which contains the details of the selection of software metrics and the error data collection. Section 3 deals with Hypothesis used for study which contains the description of ROC Analysis, the Binary Categorization results and the Ordinal Categorization results. Section 4 deals with the analysis and discussion on the applying threshold values in practice. Section 5 deals with the conclusion and future scope respectively.

2 The Experimental Design

The objective of the study is to find threshold values of software metrics that can be used to classify modules to different error categories in OO system. The following steps are followed to achieve the objective:

i. To select the software metrics.
ii. To collect the data- the metrics as well as errors data.

In this study we identify metric threshold values by analyzing the association between metrics and errors in Eclipse- a widely used industrial-strength system. We chose Eclipse because it is Open-Source System and the error data are also obtainable .Furthermore, there are several versions of Eclipse available for analysis. We collected the software metrics from three releases of Eclipse (Versions 2.0, 2.1, and, 3.0) and error data from [28, 30]. The error in Eclipse are divided into four severity categories (Nominal, Low, Medium and High) depending on the impact of errors. We conducted statistical analysis to see whether we could identify specific metrics values that could classify the Eclipse module into different error categories in two contexts:

- Binary Categorization and
- Ordinal Categorization

In binary Categorization, we investigated whether we can classify the classes into the error and no-error categories by using specific metric values. In Ordinal categorization, we again investigated whether we could classify the modules into one of the five categories (no error, nominal-impact error, low-impact error, medium-impact error and high-impact error) by using specific metric values. We belief that if we are able to classify the modules using specific metrics values, we can use these values in practice to classify modules in OO design to different error–risk categories, thus the values become thresholds for the metrics. In the following section, we present how we selected and collected the software metrics in the study.

2.1 The selection of Software Metrics

The selection of software metrics was a difficult task because there are many available metrics. We used two criteria in our selection process:

- The set of metrics cover all aspects of OO design.
- We have to be able to collect the metrics by using automated tool.

Finally, we selected 24 class level Object-Oriented metrics which are discussed in Appendix at the end of the References. These metrics cover all aspects of class level OO design due to this reason they are belonging to coupling, cohesion, inheritance, class complexity and class-size metrics. We used JHAWK [32] automated tool metric to collect these metrics from the Eclipse source code [29]. JHAWK compiled the source code and give output as each module name and their set of OO metrics. In the next section, we describe how we collected the error data.

2.2 Collection of Error Data

From [31] where Eclipse bug data set are freely available, we collected the error data from three official releases of the Eclipse project (Versions 2.0, 2.1, and 3.0) This data can be collected from version archives like CVS and bug tracking systems like BUGZILLA in two steps:

1. Identify corrections (or fixes) in version archives.
2. Use the bug tracking system to map bug reports to releases.

Pre release bug data are used for study and two types of categorization has been done on the pre release error data:

i. Binary Categorization: In this we only used two values 0 (means no error) and 1(means with error).If a class contains error in it then we put 1 in error column otherwise 0.
ii. Ordinal Categorization: In this we divide the error severity into 4 classes.

For classification our followed steps are:

A) We find the descriptive statistics of pre error data. From that we are able to know the min, different number of occurrences of error (nonzero) and max value of error data in all classes of every versions of Eclipse.

B) After that, we again find the descriptive statistics of (Min , 25% , 50% , 75% and Max) the different occurrences of number of errors (from min (nonzero) to max).

C) Based on that we classified class error data into one of five categories that are defined as follows:

- No Error: class containing zero error.
- Nominal: class containing error in the range Min<=error<25%
- Low :class containing error in the range 25%<=error<50%
- Medium: class containing error in the range 50%<=error<75%
- High: class containing error in the range 75%<=error<Max

Table 1: Descriptive statistics of error data for all different occurrences of no of errors in Eclipse (2.0, 2.1&3.0)

Version	**Min**	**25%**	**50%**	**75%**	**Max**
Eclipse 2.0	1	9	18	29	69
Eclipse2.1	1	6	12	18	24
Eclipse3.0	1	9	18	26	43

Tables 2, 3 and 4 show the descriptive statistics of all the three versions of Eclipse. It is observed that that the INTR and NSUB value was zero for at least 75% of the modules, 50% of the module having LMC value zero,25% of the module has MPC, NSUP, EXT, FOUT, COH, FIN, INST and LCOM value zero.

3 The Hypotheses

For our study, we used two contexts:

- *Binary categorization*: which classifies classes into either No-error or Error category (without differentiating error categories).
- *Ordinal categorization*: which classifies classes into four categories: No-error, Nominal, Low, Medium, and High.

The following are the Null hypotheses for our study:

• Hypothesis 1: There are no useful threshold values of OO metric that divide between the two categories of modules (the modules that had errors and those modules that did not have NO errors) in the binary categorization for the three releases. We look ahead to a threshold value for each metric that classifies classes into the Error and No-error Categories.

• Hypothesis 2: There are no useful threshold values for OO metrics that divide between any one of the error categories (Nominal, Low, Medium, and High) and the No-error category. The second hypothesis is based on our faith that software metrics can calculate module error risk level. We look ahead to threshold values (i.e., one for each error-severity level) for each metric to differentiate each Error category from the No-error category. Though we consider each Error category independently against the No-error category, we expect a some degree of order among the three error categories, i.e., Nominal≤ Low ≤ Medium ≤ High.

Table 2: Descriptive statistics for all metrics in Eclipse2.0

Metrics	Mean	Median	Std. Deviation	Minimum	Maximum	Percentiles		
						25	50	75
NOM	10.3362	6.0000	19.87652	.00	596.00	3.0000	6.0000	12.0000
LCOM	.5897	.0400	9.40281	.00	479.00	.0000	.0400	.2300
AVCC	1.8995	1.5000	1.47203	.00	26.17	1.0000	1.5000	2.4000
NOS	74.7053	29.0000	150.73286	.00	3582.00	10.0000	29.0000	78.0000
UWCS	15.5935	9.0000	35.56403	.00	1646.00	4.0000	9.0000	17.0000
INST	5.2573	2.0000	20.18810	.00	1050.00	.0000	2.0000	5.0000
PACK	7.3480	4.0000	10.25624	.00	146.00	1.0000	4.0000	9.0000
RFC	27.6640	15.0000	39.64387	.00	596.00	5.0000	15.0000	34.0000
CBO	3.6825	2.0000	4.82179	.00	76.00	1.0000	2.0000	4.0000
NLOC	98.8993	40.0000	201.74587	.00	5200.00	13.0000	40.0000	103.0000
FIN	2.1480	1.0000	4.00499	.00	74.00	.0000	1.0000	2.0000
DIT	1.8515	2.0000	.97964	.00	8.00	1.0000	2.0000	2.0000
COH	.1743	.1000	.22449	.00	1.00	.0000	.1000	.2700
LMC	2.2976	.0000	5.77804	.00	194.00	.0000	.0000	2.0000
LCOM2	83.1783	5.0000	755.51965	.00	41126.00	1.0000	5.0000	23.0000
MAXCC	4.8569	3.0000	7.77601	.00	229.00	1.0000	3.0000	6.0000
FOUT	1.8627	1.0000	3.14407	.00	69.00	.0000	1.0000	2.0000
EXT	17.3278	8.0000	26.31608	.00	325.00	.0000	8.0000	22.0000
NSUP	.8985	1.0000	1.00218	.00	7.00	.0000	1.0000	1.0000
TCC	23.8355	10.0000	48.12150	.00	1222.00	3.0000	10.0000	24.0000
NSUB	.4703	.0000	2.80751	.00	81.00	.0000	.0000	.0000
MPC	17.3278	8.0000	26.31608	.00	325.00	.0000	8.0000	22.0000
INTR	.2925	.0000	.61960	.00	7.00	.0000	.0000	.0000
CC	29.0928	13.0000	58.96999	.00	1839.00	5.0000	13.0000	31.0000

Table 3: Descriptive statistics for all metrics in Eclipse 2.1

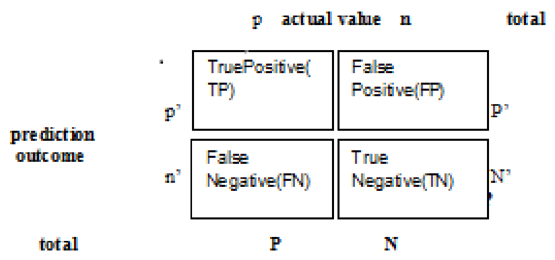
Metrics	Mean	Median	Std. Deviation	Minimum	Maximum	Percentiles		
						25	50	75
NOM	10.5589	6.0000	20.55899	.00	613.00	3.0000	6.0000	12.0000
LCOM	.5947	.0400	9.90558	.00	517.00	.0000	.0400	.2400
AVCC	1.9348	1.5300	1.53548	.00	30.50	1.0000	1.5300	2.4475
NOS	77.9879	30.0000	156.41658	.00	3592.00	10.0000	30.0000	80.0000
UWCS	15.9690	9.0000	36.42494	.00	1740.00	4.0000	9.0000	18.0000
INST	5.4101	2.0000	20.50783	.00	1127.00	.0000	2.0000	5.0000
PACK	7.6482	4.0000	10.92447	.00	151.00	1.0000	4.0000	9.0000
RFC	29.1601	15.0000	42.10427	.00	613.00	5.0000	15.0000	36.0000
CBO	3.6726	2.0000	4.88056	.00	76.00	1.0000	2.0000	4.0000
NLOC	103.3093	42.0000	209.97243	.00	5221.00	14.0000	42.0000	107.0000
FIN	2.1461	1.0000	4.08431	.00	75.00	.0000	1.0000	2.0000
DIT	1.8420	2.0000	.96413	.00	8.00	1.0000	2.0000	2.0000
COH	.1783	.1100	.22668	.00	1.00	.0000	.1100	.2800
LMC	2.4737	.0000	6.13886	.00	195.00	.0000	.0000	3.0000
LCOM2	90.1634	5.0000	769.33205	.00	43119.00	1.0000	5.0000	25.0000
MAXCC	5.0193	3.0000	8.14865	.00	229.00	1.0000	3.0000	6.0000
FOUT	1.8365	1.0000	3.07557	.00	69.00	.0000	1.0000	2.0000
EXT	18.6012	8.0000	28.29340	.00	335.00	1.0000	8.0000	24.0000
NSUP	.8855	1.0000	.98587	.00	7.00	.0000	1.0000	1.0000
TCC	24.8016	10.0000	50.03921	.00	1226.00	3.0000	10.0000	26.0000
NSUB	.4471	.0000	2.74776	.00	81.00	.0000	.0000	.0000
MPC	18.6012	8.0000	28.29340	.00	335.00	1.0000	8.0000	24.0000
INTR	.2962	.0000	.64061	.00	9.00	.0000	.0000	.0000
CC	25.0977	10.0000	50.06080	.00	1226.00	4.0000	10.0000	26.0000

Table 4: Descriptive statistics for all metrics in Eclipse3.0

Metrics	Mean	Median	Std. Deviation	Minimum	Maximum	Percentiles		
						25	50	75
NOM	10.2326	6.0000	20.35503	.00	845.00	3.0000	6.0000	12.0000
LCOM	.6368	.0400	10.15964	.00	646.00	.0000	.0400	.2500
AVCC	1.9265	1.5000	1.56149	.00	42.67	1.0000	1.5000	2.4500
NOS	76.0767	28.0000	158.08140	.00	3614.00	9.0000	28.0000	78.0000
UWCS	15.3086	8.0000	35.92773	.00	1950.00	4.0000	8.0000	17.0000
INST	5.0760	2.0000	19.94747	.00	1257.00	.0000	2.0000	5.0000
PACK	7.8631	4.0000	11.36715	.00	204.00	1.0000	4.0000	10.0000
RFC	28.7772	15.0000	42.75694	.00	847.00	5.0000	15.0000	35.0000
CBO	3.6007	2.0000	5.07228	.00	110.00	1.0000	2.0000	4.0000
NLOC	100.1076	40.0000	205.90128	.00	4879.00	13.0000	40.0000	105.0000
FIN	2.1212	1.0000	4.16362	.00	109.00	.0000	1.0000	2.0000
DIT	1.7698	2.0000	.93040	.00	8.00	1.0000	2.0000	2.0000
COH	.1799	.1000	.23473	.00	1.00	.0000	.1000	.2800
LMC	2.4447	.0000	5.89691	.00	188.00	.0000	.0000	3.0000
LCOM2	84.2403	5.0000	748.85147	.00	54396.00	1.0000	5.0000	22.0000
MAXCC	4.9551	3.0000	8.27738	.00	244.00	1.0000	3.0000	6.0000
FOUT	1.7975	1.0000	3.43117	.00	98.00	.0000	1.0000	2.0000
EXT	18.5446	8.0000	29.03112	.00	382.00	.0000	8.0000	23.0000
NSUP	.8086	1.0000	.93752	.00	7.00	.0000	1.0000	1.0000
TCC	24.3188	10.0000	51.06310	.00	1399.00	3.0000	10.0000	25.0000
NSUB	.3972	.0000	2.66535	.00	89.00	.0000	.0000	.0000
MPC	18.5446	8.0000	29.03112	.00	382.00	.0000	8.0000	23.0000
INTR	.3203	.0000	.67277	.00	9.00	.0000	.0000	.0000
CC	24.6391	10.0000	51.21183	.00	1405.00	3.0000	10.0000	25.0000

3.1 THE ROC ANALYSIS

ROC is a diagnostic accuracy test [37]. The ROC method can be used to assess the quality of the information provided by the classification of classes into a binary category using a metric. To plot the ROC curve, we need to define two variables: one binary (i.e., 0 or 1) and another continuous. In our study, we have two contexts: the binary and the ordinal categorization. The classes in the ordinal categorization should be considered one by one, i.e., we need to plot the ROC curve for each category (Nominal, Low, Medium, and High) leaving the No-error category as the option. The continuous variable in both categorizations is the metric used in the study._ There are four possible outcomes from a binary classifier. If the outcome from a prediction is ***p*** and the actual value is also ***p***, then it is called a ***true positive (TP)***; however if the actual value is ***n*** then it is said to be a ***false positive (FP)***. Conversely, a ***true negative (TN)*** has occurred when both the prediction outcome and the actual value are ***n***, and ***false negative (FN)*** is when the prediction outcome is ***n*** while the actual value is ***p***. From ***P*** positive instances and ***N*** negative instances. The four outcomes can be formulated in a 2×2 contingency table or confusion matrix, as follows:



		actual value p	actual value n	total
prediction outcome	p'	TruePositive(TP)	False Positive(FP)	P'
	n'	False Negative(FN)	True Negative(TN)	N'
total		P	N	

Table V: Confusion Matrix

To draw a ROC curve, only the ***true positive rate (TPR)*** and ***false positive rate (FPR)*** are needed (as functions of some classifier parameter). The ***TPR*** defines how many correct positive results occur among all positive samples available during the test. ***FPR***, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

A ROC space is defined by ***FPR*** and ***TPR*** as ***x*** and ***y*** axes respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since ***TPR*** is equivalent with sensitivity and ***FPR*** is equal to **1 − specificity**, the ROC graph is sometimes called the **sensitivity vs. (1 − specificity)** plot. Each prediction result or instance of a confusion matrix represents one point in the ROC space.

The best possible prediction method would yield a point in the upper left corner or coordinate (0, 1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives).

The sensitivity and specificity are calculated from the confusion matrix as follows:

Sensitivity=***tp rate***

=TP/P

Specificity=***1−fp rate***

=1−FP/N

We need a criterion to choose a threshold value for a metric (sensitivity, 1-specficty pair) to balance between benefits and costs. Our choice is a commonly used criterion that chooses the pair that has the maximum value for both sensitivity and specificity [38]. In other words, we want to minimize false-positives (false alarms) and false-negatives at the same time. The threshold values obtained from the ROC analysis need to be validated by the classification performance of the ROC before they can be used in practice. The area under ROC curve ranges between 0 and 1—it measures the classification performance of using the threshold value to put classes into Error (flag alarm) or No-error (don't flag alarm) categories. The graph below shows three ROC curves representing excellent, good, and worthless tests plotted on the same graph. The accuracy of the test depends on how well the test separates the group being tested into those with and without the error in classes. Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test. A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system:

The general rule to evaluate the classification performance is to find the area under the curve (AUC)[38]:

• AUC=0.5 means no good classification;

• 0.5<AUC<0.6 means poor classification;

• 0.6≤AUC<0.7 means fair classification;

• 0.7≤AUC<0.8 means acceptable classification;

• 0.8≤AUC<0.9 means excellent classification;

• AUC≥0.9 means outstanding classification.

For calculating AU, we have used IBM SPSS statistics Version 19. The rationale behind the classification of false positive as false alarm is that some classes showing not so good result of sensitivity but good result for specificity value. It means the threshold value obtained corresponding for this value from AUC not give the right threshold value for detecting error free or error prone classes. So, our aim is to take those pair of sensitivity and specificity value that has higher value of sensitivity (i.e. true positive rate) and less specificity (i.e. false positive rate).

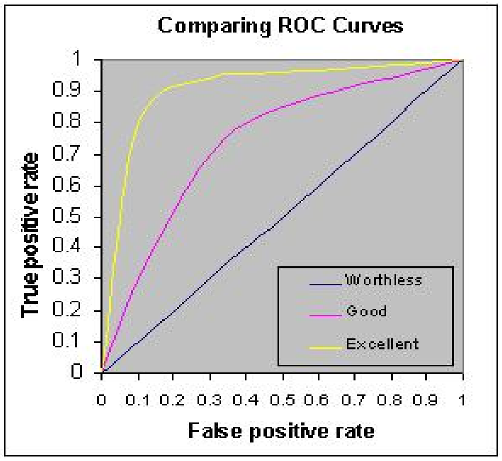


Figure1: Compairing ROC curve for calculating AUC

The practical threshold values should have a classification performance falls at least within the acceptable range. Therefore, the metrics that have AUC within the acceptable (or higher) range will be considered valid; otherwise, we conclude that we could not find a practical threshold value for the metric. The ROC analysis is very effective for data with skewed distribution and unequal classification error costs [39] and is suitable for analyzing our data because our data is not normally distributed and somewhat skewed.

3.2 The Binary Categorization Results

Table 6 shows that the AUC for all metrics in the three releases falls below the acceptable range. We consider AUC as an indicator of practical and useful threshold values. We conclude that the threshold values presented in Table VI are not practical, because the classification power as represented by the AUC is either fair or poor. So, we cannot reject Hypothesis 1 (i.e., There is no significant classification in the metric between the two categories of modules). Therefore, we could not find practical threshold values for the metrics to differentiate Error and No-error classes.

We conclude that the threshold values presented in Table VI are only practical for NOS, PACK, RFC, NLOC, EXT MPC and CC metrics only for Eclipse2.1 and Eclipse 3.0 not for Eclipse2.0. Other metric thresholds are not practical, because the classification power represented by the AUC is either fair or poor. So, we cannot fully accept or reject Hypothesis 1 (i.e., There is no significant classification in the metric between the two categories of modules). Therefore, we could not find practical threshold values for the metrics to differentiate Error and No-error classes.

3.3 The Ordinal Categorization Results

For testing Hypothesis 2, we tried the same experiment on more fine-grained error categories (Nominal, Low, Medium, and High). The identified threshold values are presented in Table 7. In all the three releases, we noticed that the metric threshold values for the nominal category were either poor or fair. The threshold values for the low, Medium and High categories for the NOS, UWCS, CC, RFC, NLOC, EXT, MPC, LMC, TCC, PACK, NOM, LCOM2, INST, CBO, MAXCC, FOUT and AVCC metrics were valid, practical, and useful, because their AUCs were within the acceptable and excellent range, whereas the AUCs of the other metrics were neither practical nor useful, because they were out of the acceptable range. Therefore we concluded that Hypothesis 2 was rejected for the NOS, UWCS, CC, RFC, NLOC, EXT, MPC, LMC, TCC, PACK, NOM, LCOM2, INST, CBO, MAXCC, FOUT and AVCC metrics. Thus, we can use the threshold values for these metrics.

Table 6: Threshold recognized based on the binary categorization

Metrics Used	Eclipse 2.0		Eclipse 2.1		Eclipse3.0	
	AUC	Threshold	AUC	Threshold	AUC	Threshold
NOM	0.639	7.500	0.668	6.500	0.686	6.500
LCOM	0.540	.0450	0.541	0.0450	0.560	0.0450
AVCC	0.636	1.615	0.675	1.7450	0.660	1.6250
NOS	0.663	31.500	**0.714**	38.500	**0.717**	37.500
UWCS	0.619	8.500	0.664	10.500	0.689	9.500
INST	0.561	1.500	0.612	1.500	0.644	1.500
PACK	0.673	3.500	**0.727**	4.500	**0.706**	4.500
RFC	0.680	15.500	**0.720**	19.500	**0.719**	18.500
CBO	0.587	2.500	0.578	2.500	0.589	1.500
NLOC	0.663	47.500	**0.712**	52.500	**0.717**	51.500
FIN	0.513	0.500	0.507	0.500	0.537	0.500
DIT	0.534	1.500	0.537	1.500	0.522	1.500
COH	0.541	.0450	0.550	0.1050	0.539	0.1050
LMC	0.642	0.500	0.678	0.500	0.675	0.500
LCOM2	0.624	5.500	0.625	5.500	0.656	5.500
MAXCC	0.647	2.500	0.692	3.500	0.686	3.500
FOUT	0.620	1.500	0.609	0.500	0.615	0.500
EXT	0.681	10.500	**0.726**	11.500	**0.715**	10.500
NSUP	0.534	0.500	0.533	0.500	0.521	0.500
TCC	0.658	9.500	0.697	12.500	**0.705**	12.500
NSUB	0.502	0.500	0.514	0.500	0.511	0.500
MPC	0.681	8.500	**0.726**	11.500	**0.715**	10.500
INTR	0.521	0.500	0.536	0.500	0.548	0.500
CC	0.645	10.500	0.696	15.500	**0.709**	15.500

4 Analysis and Discussion on Applying Thresholds in Practice

For the binary category, we could not identify useful and practical threshold values to separate classes into either erroneous or not-erroneous classes. For the ordinal category, we identified useful and practical threshold values for the NOS, UWCS, CC, RFC, NLOC, EXT, MPC, LMC, TCC, PACK, NOM, LCOM2, INST, CBO, MAXCC, FOUT and AVCC metrics. We summarized the identified threshold values in Table VIII for both the Medium and the High categories. Our expectation was that the threshold values in the High category should be higher than that in the Low and Medium category. We observed this behavior for the Eclipse 2.0 and 2.1version, whereas the Versions 3.0 data showed the opposite (i.e., low and Medium values are larger than High values). This result indicates that our Low, Medium and High categories are not so distinguishable. The first reading of it is that this was caused by the abnormal distribution of error categories. But, for abnormal distributed data the ROC analysis is effective [32]. As these categories were ordinal rankings, we merged the Low, Medium and High categories into one category by finding the average of all these three categories of every version and again recalculated threshold values for these metrics as shown in Table 9. We used the Sensitivity and Specificity values (they indicate efficiency in classifying faulty classes) to order the metric values in Table IX. We noticed that the size metrics (NOS and UWCS) came before the (CC and RFC) metrics. This information tells that the size metrics are better indicators of faulty classes. These results showed that the threshold values differed from one release to another. With the highest Sensitivity value as the selection standard, we choose the final threshold values for the NOS, UWCS, CC, RFC, NLOC, EXT, MPC, LMC, TCC, PACK, NOM, LCOM2, INST, CBO, MAXCC, FOUT and AVCC metrics and result is summarized in Table 10. These values can be used by developers as a guideline for designing classes, if the metrics exceed the threshold value then, there is chances of error prone classes.

Table 7. Threshold values for all metrics

METRIC	ECLIPSE 2.0								ECLIPSE 2.1								ECLIPSE 3.0							
USED	NOMINAL		LOW		MID		HIGH		NOMINAL		LOW		MID		HIGH		NOMINAL		LOW		MID		HIGH	
	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR.	AUC	THR
NOM	0.64	5.50	0.75	9.50	0.76	10.50	0.87	11.50	0.64	6.50	0.81	11.50	0.85	12.50	0.81	13.50	0.67	6.50	0.83	11.50	0.77	8.50	0.72	9.50
LCOM	0.54	0.05	0.53	0.35	0.51	0.03	0.60	0.08	0.54	0.05	0.54	0.05	0.56	0.05	0.40	0.04	0.56	0.05	0.50	0.03	0.53	0.01	0.39	0.01
AVCC	0.64	1.57	0.67	1.71	0.65	1.50	0.69	1.99	0.66	1.68	0.72	2.11	0.71	2.20	0.70	2.17	0.65	1.71	0.73	2.11	0.62	1.86	0.68	1.99
NOS	0.66	30.50	0.78	32.50	0.81	73.00	0.89	77.50	0.69	38.50	0.84	88.50	0.84	99.50	0.90	123.50	0.70	37.50	0.84	82.50	0.78	74.50	0.75	70.50
UWCS	0.62	8.50	0.74	10.50	0.80	17.50	0.88	16.50	0.64	9.50	0.82	16.50	0.83	15.50	0.96	24.50	0.68	9.50	0.84	17.50	0.78	14.50	0.78	18.50
INST	0.56	1.50	0.67	3.50	0.77	3.50	0.81	4.50	0.54	1.50	0.71	3.50	0.73	3.50	0.98	17.50	0.63	1.50	0.79	4.50	0.74	3.50	0.79	4.50
PACK	0.67	3.50	0.83	7.50	0.83	9.50	0.87	12.50	0.71	4.50	0.80	8.50	0.78	8.50	0.92	12.50	0.70	4.50	0.77	7.50	0.69	5.00	0.70	6.50
RFC	0.68	15.50	0.80	29.50	0.80	33.50	0.90	40.50	0.69	19.50	0.85	38.50	0.85	42.50	0.86	49.50	0.71	18.50	0.85	39.50	0.73	24.50	0.74	31.50
CBO	0.59	1.50	0.63	2.50	0.81	4.50	0.92	4.50	0.56	2.50	0.70	2.50	0.76	3.50	0.81	2.50	0.58	1.50	0.76	3.50	0.70	3.50	0.75	3.50
NLOC	0.66	41.50	0.77	57.50	0.81	83.50	0.89	92.50	0.69	52.50	0.83	114.50	0.83	129.50	0.88	163.50	0.71	51.50	0.84	111.50	0.77	69.50	0.75	95.50
FIN	0.51	0.50	0.53	1.50	0.70	1.50	0.74	1.50	0.50	0.50	0.59	0.50	0.67	0.50	0.60	0.50	0.53	0.50	0.68	1.50	0.70	1.50	0.68	1.50
DIT	0.53	1.50	0.52	1.50	0.52	1.50	0.58	1.50	0.54	1.50	0.50	1.50	0.43	0.50	0.42	1.50	0.52	1.50	0.52	1.50	0.41	0.50	0.44	1.50
COH	0.54	0.95	0.50	1.45	0.46	0.05	0.53	0.05	0.55	0.11	0.49	0.07	0.48	0.55	0.41	0.65	0.54	0.11	0.45	0.06	0.45	0.35	0.40	0.03
LMC	0.64	0.50	0.76	0.50	0.79	1.50	0.90	2.50	0.65	0.50	0.83	2.50	0.81	1.50	0.87	3.50	0.66	0.50	0.82	2.50	0.70	0.50	0.78	1.50
LCOM2	0.62	5.50	0.73	9.50	0.79	13.50	0.90	42.50	0.60	5.50	0.75	15.50	0.83	27.50	0.83	25.50	0.64	5.50	0.84	27.50	0.72	10.50	0.73	18.50
MAXCC	0.65	2.50	0.72	4.50	0.72	4.50	0.80	5.50	0.67	3.50	0.78	5.50	0.80	6.50	0.73	5.50	0.68	3.50	0.79	5.50	0.69	4.50	0.72	4.50
FOUT	0.62	0.50	0.65	1.50	0.71	1.50	0.85	2.50	0.59	0.50	0.72	1.50	0.76	1.50	0.88	2.50	0.60	0.50	0.75	1.50	0.65	0.50	0.75	1.50
EXT	0.68	8.50	0.81	16.50	0.80	22.50	0.90	31.50	0.70	11.50	0.84	25.50	0.83	26.50	0.89	33.50	0.70	10.50	0.83	23.50	0.69	12.50	0.75	20.50
NSUP	0.53	0.50	0.53	0.50	0.53	0.50	0.60	0.50	0.54	0.50	0.50	0.50	0.46	0.50	0.48	0.50	0.52	0.50	0.52	0.50	0.42	0.50	0.47	0.50
TCC	0.66	8.50	0.75	18.50	0.77	20.50	0.88	31.50	0.67	12.50	0.82	23.50	0.85	31.50	0.81	36.50	0.69	12.50	0.84	26.50	0.77	21.50	0.73	22.50
NSUB	0.50	0.50	0.53	0.50	0.55	0.50	0.49	0.50	0.51	0.50	0.53	0.50	0.57	0.50	0.45	0.50	0.51	0.50	0.51	0.50	0.56	0.50	0.50	0.50
MPC	0.68	8.50	0.81	16.50	0.80	12.50	0.90	31.50	0.70	11.50	0.84	25.50	0.83	26.50	0.89	33.50	0.70	10.50	0.83	23.50	0.69	12.50	0.75	20.50
INTR	0.52	0.50	0.62	0.50	0.62	0.50	0.70	0.50	0.53	0.50	0.59	0.50	0.70	0.50	0.66	0.50	0.55	0.50	0.56	0.50	0.67	0.50	0.56	0.50
CC	0.65	10.50	0.75	14.50	0.80	25.50	0.88	36.50	0.67	15.00	0.83	30.50	0.84	38.50	0.93	45.50	0.70	15.50	0.84	31.50	0.77	24.50	0.76	26.50

As we did our study on Eclipse, we believe that the research results can be generalized to the OO systems that are similar to Eclipse—an industrial-strength system that is continuously evolving with thousands of classes. Our idea is based on the fact that we analyzed the system at the class level and drew the conclusions from the classes, not from the system. Any OO system that is as complex as Eclipse shares (among its class structures) common design attributes such as size, inheritance, objects, message passing, abstract data types, and polymorphism[25]; these are the attributes that the metrics measure. The fact that Eclipse is an open-source system does not limit our research results to just the open-source systems because the open-source model adopted by Eclipse is different from that adopted by Linux. The Eclipse evolution is managed by a centralized team of engineers (in IBM Corporation). In this regard, the evolution of Eclipse is similar to many large industrial software systems. It is reasonable to believe that the software engineering process that controls the evolution of the Eclipse project is similar to the processes used by other organizations to evolve software systems that are similar in size and complexity to Eclipse.

Table 9. Candidate threshold values and their distributions

METRIC USED	VER.	LOW THR.	MEDIUM THR.	HIGH THR.
NOM	E2.0	9.5	10.50	11.5
	E2.1	11.5	12.50	13.5
	E3.0	11.5	8.50	9.5
AVCC	E2.0	N/A	N/A	N/A
	E2.1	2.1050	2.1950	2.1650
	E3.0	2.1050	N/A	N/A
NOS	E2.0	32.5	73	77.5
	E2.1	88.5	99.5	123.5
	E3.0	82.5	74.5	70.50
UWCS	E2.0	10.5	17.5	16.5
	E2.1	16.5	15.4	24.5
	E3.0	17.5	14.5	18.5
INST	E2.0	N/A	3.5	4.5
	E2.1	3.5	3.5	17.5
	E3.0	4.5	3.5	4.5
PACK	E2.0	7.5	9.5	12.5
	E2.1	8.5	8.5	12.5
	E3.0	7.5	N/A	6.5
RFC	E2.0	29.5	33.5	40.5
	E2.1	38.5	42.5	49.5
	E3.0	39.5	24.5	31.5
CBO	E2.0	N/A	4.5	4.5
	E2.1	N/A	3.5	2.5
	E3.0	3.5	3.5	3.5
NLOC	E2.0	57.5	83.5	92.5
	E2.1	114.5	129.5	163.5
	E3.0	111.5	69.5	95.5
FIN	E2.0	N/A	1.500	1.500
	E2.1	N/A	N/A	N/A
	E3.0	N/A	1.500	N/A
LMC	E2.0	0.500	1.500	2.500
	E2.1	2.500	1.500	3.500
	E3.0	2.500	N/A	1.500
LCOM2	E2.0	9.500	13.500	42.500
	E2.1	15.500	27.500	25.500
	E3.0	27.5	10.5	18.5
MAXCC	E2.0	4.5	4.5	5.5
	E2.1	5.5	6.5	5.5
	E3.0	5.5	N/A	4.5
FOUT	E2.0	N/A	1.5	2.5
	E2.1	1.5	1.5	2.5
	E3.0	1.5	N/A	1.5
EXT	E2.0	16.5	22.5	31.5
	E2.1	25.5	26.5	33.5
	E3.0	23.5	N/A	20.5
TCC	E2.0	18.5	20.5	31.5
	E2.1	23.5	31.5	36.5
	E3.0	26.5	21.5	22.5
MPC	E2.0	16.5	12.5	31.5
	E2.1	25.5	26.5	33.5
	E3.0	23.5	N/A	20.5
CC	E2.0	14.5	25.5	36.5
	E2.1	30.5	38.5	45.5
	E3.0	31.5	24.5	26.5

Table 10. Candidate threshold values.

Metrics	Threshold	Rank of use
NOS	61	1
UWCS	119	2
CC	26	3
RFC	44	4
NLOC	78	5
EXT	29	6
MPC	29	7
LMC	2	8
TCC	31	9
PACK	10	10
NOM	13	11
LCOM2	22	12
INST	8	13
CBO	4	14
MAXCC	6	15
FOUT	2	16
AVCC	2	17

4.1 Discussion

The aim of this work is to test the two hypothesis defined in hypothesis section .The results shows that there are no acceptable threshold values for each Eclipse version of the OO metrics that separate between the two categories of modules(the modules that had errors and modules that did not have errors) in the binary categorization. We found the threshold value for some metrics NOS, PACK, RFC, NLOC, EXT,MPC for Eclipse 2.1 and Eclipse 3.0, but not for Eclipse 2.0. That's why we cannot reject the Hypothesis 1. On the other hand , in case of ordinal categorization we got the threshold value for some metrics NOS, UWCS, CC, RFC, NLOC, EXT, MPC, LMC, TCC, PACK,NOM,LCOM2,INST,CBO,MAXCC,FOUT,AVCC for the merged categories(Low, Medium and High categories), but not for the nominal category. Therefore, we rejected Hypothesis 2.

Table 11: Comparison of threshold

Metrics	Rosenberg threshold	Shatnawi et. al. threshold	Proposed threshold
RFC	100	44	44
CBO	5	13	4

We compared the threshold values that we identified, which are tabulated in Table 11, with the threshold values suggested by [40] and [20]. We noticed that our threshold values are all smaller than theirs (only two metrics are common). As our assumption is that there are more errors in the modules before the release so that smaller threshold values should be able to identify error prone modules in the development phase. The result is according to our requirement. We then conducted a test on their values by applying their threshold values on the Eclipse system and calculated the Sensitivity and 1-specificity. The test result shows that Rosenberg's thresholds are not useful but Shatnawi et al threshold are somewhat useful but not totally useful in finding faulty classes in the Eclipse system, i.e., the sensitivity values were low.

5 Conclusions and Future Work

Eclipse data (bug and metric data) and Statistical analysis (ROC and AUC) are used to test the threshold values of OO metrics in two context, the binary (Error and No-error) and the ordinal .We have used total 24 software metrics out of which 17 metrics threshold value are successfully identified which can distinguish between high risk error prone class in the ordinal categorization from the No-error classes. We were unable to find threshold values for the metrics in binary categorization. We believe that our research findings are useful for software engineers because our approach help them to easily estimate the metric values for the classes that they design and our threshold value give them idea not to move their design into a high-risk area. The results are beginning because we only validated the values using three releases of one system that is in the pre-release evolution process. On the other hand, we consider that the findings are a good step because the values are explicitly associated with a concerned design factor: the error proneness of Java classes. In terms of future scope we suggest more empirical studies on this subject, particularly on how effective these values are in various contexts.The thresholds should be applied to other software systems for verifying the correctness. Since the verification is missing,the thresholds are of limited practical value at this stage.We will apply these threshold value in future for different software systems.

References

[1] Kecia A. M. Ferreira, Mariza A. S. Bigonha, Roberto S. Bigonha, Luiz F. O. Mendes and Heitor C. Almeida, (February, 2012). Identifying thresholds for object-oriented software metrics', Journal of Systems and Software, Volume 85 No. 2, pp. 244-257.

[2] Myers G, Badgett T, Thomas T, and Sandler C. (2004). *The Art of Software Testing*, 2nd ed,,Wiley, Hoboken NJ.

[3] Basili V, Briand L, and Melo W. (1996). A validation of object-oriented metrics as quality indicators, *IEEE Transactions on Software Engineering*; volume 22 No.10, pp. 751–761.

[4] Briand L, Daly J and Wust J. (1998). A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering*; Volume 3 No.1, pp. 65–117.

[5] Gyimothy T, Ferenc R and Siket I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction, *IEEE Transactions on Software Engineering*; Volume 31 No.10, pp. 897–910.

[6]Shatnawi R and Li W. (2008). The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process, *Journal of Systems and Software*, Volume 81 No.11, pp. 1868–1882.

[7] Subramanyam R, and Krishnan M. (2003). Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects, *IEEE Transactions on Software Engineering*, Volume 29 No.4, pp. 297–310.

[8] Victor R. Basili, Lionel Briand and Walcélio L. Melo (April 1995). *A Validation of Object-Oriented Design Metrics as Quality Indicators*, Univ. of Maryland, Dep. of Computer Science, College Park, MD, 20742 USA/REP.

[9] R. A. Khan, K. Mustafa and S. I. Ahson. (2007). An Empirical Validation of Object Oriented Design Quality Metrics', *Comp. & Info. Sci*., J. King Saud Univ., Vol. 19, pp. 1-16.

[10] Saïda Benlarbi and Walcelio L. Melo (1999). Polymorphism Measures for Early Risk Prediction', *Proceedings of the 1999 International Conference on Software engineering*, Page(s): 334 – 344.

[11] Binkley, A.B. and Schach, S.R (1998). Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures, P*roceedings of the 1998 International Conference on Software Engginеering*, Page(s): 452 - 455 .

[12] Lionel Briand, Prem Devanbu and Walcelio Melo (1997) An Investigation into Coupling Measures for C++', *Proc. of the 19th International Conference on Software Engeneering*, 18-23 May, 1997.

[13] Lionel C. Briand, Jürgen Wust, Stefan Ikonomovski and Hakim Lounis (1998). *A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study* ,ISERN-98-29, IESE-Report No. 47.98/E.

[14] L.C..Briand, J.Daly and V.Porter, (2000). Exploring the Relationships between Design Measu res and Software Quality in Object-Oriented Systems, *Journal of Systems and software*,Volume 51, pp. 245—273.

[15] Brito e Abreu, F. and Melo, W.(1996)'Evaluating the impact of object-oriented design on software quality, *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, pp. 90 – 99.

[16] Cartwright, M. and Shepperd, M. (2000). An empirical investigation of an object-oriented software system, *Software Engineering, IEEE Transactions on* , pp.786 - 796 .

[17] Harrison, R., Counsell, S. and Nithi, R. (1998). Coupling metrics for object-oriented design, *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International*, pp. 150 – 157.

[18] Mei-Huei Tang , Ming-Hung Kao and Mei-Hwa Chen / (1999). An empirical study on object-oriented metrics, *Proceedings of the Sixth International Software Metrics Symposium, 1999,* pp. 242 – 249.

[19] Shatnawi, R. (2010). A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems, *IEEE Transactions on Software Engineering*, Volume 36 No. 2 ,pp. 216 – 225.

[20] Raed Shatnawi, Wei Li,James Swain and Tim Newman. (2010). Finding software metrics threshold values using ROC curves, *Journal of Software Maintenance and Evolution: Research and Practice*, Volume 22No. 1, pp. 1–16.

[21] Raed Shatnawi and Qutaibah Althebyan (2013). An Empirical Study of the Effect of Power Law Distribution on the Interpretation of OO Metrics, *ISRN Software Engineering*, Volume 2013 (2013), Article ID 198937, 18 pages.

[22] Sarabjit Kaur, Satwinder Singh and Harshpreet Kaur (2013). A Quantitative Investigation Of Software Metrics Threshold Values At Acceptable Risk Level, *International Journal of Engineering Research & Technology (IJERT)*, Vol. 2 ,Issue 3, ISSN: 2278-0181.

[23] M.Lorenz and J.Kidd (1994). *Object- Oriented Software Metrics*, Prentice –Hall.

[24] Gyimothy T, Ferenc R and Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction, *IEEE Transactions on Software Engineering* 2005, Volume 31 No.10, pp. 897–910.

[25] Subramanyam R and Krishnan M. (2003). Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects', *IEEE Transactions on Software Engineering* 2003,Volume 29 No. 4, pp. 297–310.

[26] Li W and Shatnawi R. (2007). An empirical study of the bad smells and errors in object-oriented design, *Journal of Systems and Software* 2007; Volume 80 No. 7, pp. 1120–1128.

[27]Yogesh Singh, Arvinder Kaur and Ruchika Malhotra (2008). Emprical investigation to find the Effect of Design Metrics on Fault Proneness, *Proceedings of the 2nd national conference*;INDIACOM-2008

[28] Eclipse bug data (for archived releases): http://www.st.cs.uni-sb.de/softevo/bug-data/eclipse (Accessed 20 November 2012)

[29] Eclipse source code (for archived releases): http://archive.eclipse.org/eclipse/downloads/ (Accessed 3 December 2012)

[30] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller. (2006). If your bug database could talk..., in *Proceedings of the 5th International Symposium on Empirical Software Engineering. Volume II: Short Papers and Posters*, 2006, pp. 18-20.

[31] BalaSubramanian N. V. (1996). Object-Oriented Metrics, *Asian Pacific Software Engineering Conference* (APSEC-96), pp. 30-34.

[32] JHAWK metrics reference http://www.virtualmachinery.com/jhawkreferences.html (Accessed March 2013)

[33] Chidamber S and Kemerer C. (1994). A metrics suite for object oriented design', *IEEE Transactions on Software Engineering* 1994, Volume 20 No.6, pp. 476–493.

[34] JHAWK metrics reference http://www.aivosto.com/project/help/pm-oomisc.html (Accessed march 2013)

[35] T. J. McCabe (1976). A complexity measure, *IEEE Trans. Software Eng*., Volume 2 No. 4, pp. 308–320.

[36] R. Barker, and E. Tempero, (2007). A Large-Scale Empirical Comparison of Object-Oriented Cohesion Metrics, *In Proceedings of the 14th Asia-Pacific Software Engineering Conference*, 2007, pp. 414-421.

[37] Zweig M and Campbell G. (1993). Receiver-operating characteristic (ROC) plots: A fundamental evaluation tool in clinical medicine, *Clinical Chemistry* 1993; 39(4):561–577.

[38] Hosmer D and Lemeshow S. (2000). *Applied Logistic Regression*, Wiley-Interscience, 2nd ed,,New York NY.

[39] Fawcett T. (2004). *ROC graphs: Notes and practical considerations for researchers*, Technical Report, HP Laboratories, Page Mill Road, Palo Alto, CA, 2004; 38.

[40] Rosenberg L. (1998). Applying and interpreting object oriented metrics', *Software Technology Conference*, 1998.