A MODEL TO COMPARE THE DEGREE OF REFACTORING OPPORTUNITIES OF THREE PROJECTS USING A MACHINE ALGORITHM

Gauri Khurana[1] and Sonika Jindal[2]

[1] Student, Department of Computer Science, Shaheed Bhagat Singh State Technical Campus, Ferozepur.
Punjab Technical University, Jalandhar
khurana.gauri@gmail.com
[2]Assistant Professor, Department of Computer Science, Shaheed Bhagat Singh State Technical Campus, Ferozepur.
Punjab Technical University, Jalandhar
sonika.manoj@gmail.com

ABSTRACT

Refactoring is applied to the software artifacts so as to improve its internal structure, while preserving its external behavior. Refactoring is an uncertain process and it is difficult to give some units for measurement. The amount to refactoring that can be applied to the source-code depends upon the skills of the developer. In this research, we have perceived refactoring as a quantified object on an ordinal scale of measurement. We have a proposed a model for determining the degree of refactoring opportunities in the given source-code. The model is applied on the three projects collected from a company. UML diagrams are drawn for each project. The values for source-code metrics, that are useful in determining the quality of code, are calculated for each UML of the projects. Based on the nominal values of metrics, each relevant UML is represented on an ordinal scale. A machine learning tool, weka, is used to analyze the dataset, imported in the form of arff file, produced by the three projects.

KEYWORDS

Refactoring, Refactoring Opportunities, Source-code metrics, UML diagrams, Ordinal scale, Weka

1. INTRODUCTION

Refactoring means to re-influence, something that already exists [1]. Whenever we make a change in the code, we are refactoring the software. Refactoring was introduced by W.F. Opdyke in his dissertation and the further studies were carried out by other researchers. Polymorphism has some similar properties as refactoring. We apply polymorphism to re-use the given statement for performing different functions. Thus polymorphism re-influences the already existing code. Refactoring is also done so as to make the software reusable. Refactoring is done to simplify complicated methods. A major challenge while refactoring the code is to preserve the external behavior of the software [2]. It is important to run a test suite before as well as after applying refactoring to check whether the original behavior of the software is maintained. Sometimes a small change in the source-code can alter the original behavior of the software. A very small change to a software system is much more prone to error than larger changes, because people tend to take very small changes less seriously.

The process of refactoring is uncertain and imprecise. The number of refactoring that can be applied to the software depends upon the dexterity of the developer. Integrated Development Environments (IDE's) such as Eclipse 3.5, NetBeans 6.7, IntelliJ IDEA 8.1, Visual Studio 2008, and Refactor! Pro 2.5, etc offer support for refactoring [3]. Refactoring by hand has long been assumed to be error-prone. In order to help developers perform efficient and correct refactoring, various refactoring tools have been developed. These tools promise to help developers refactor faster and with a smaller probability of introducing defects. Despite the wide availability, refactoring tools are underused; according to two case studies, about 90% of refactorings are performed by hand [4]. To address this issue, [5] conducted a formative study of developers' manual refactoring process, suggesting that developers' reliance on "chasing error messages" when manually refactoring is an error-prone manual refactoring strategy. Additionally, their study distilled a set of manual refactoring workflow patterns. Using these patterns, they designed a novel refactoring tool called BeneFactor. The IDE's determine which refactoring/s is/are applicable based on the selection of a code fragment by the developer. We have used 3 projects collected from a company to study the proposed model. The evaluation of this research work is done in four parts. In the first part, the UML diagrams are drawn for the three projects and the values of source-code metrics are calculated for each UML diagram. In the second step, based on the values of metrics, UMLs are represented on an ordinal scale of measurement. In The third step, statistical analysis of the dataset, produced in the previous step, is done based on the Naive Bayes algorithm with the help of a machine learning tool 'weka'. In the last part of the research, the results obtained through statistical analysis are discussed for acceptance or rejection of the refactoring opportunities.

2. RELATED WORK

Nikolaos and Alexander in [3] have proposed a technique that extracts refactoring suggestions introducing polymorphism. Polymorphism is one of the most important features offered by object-oriented programming languages, since it allows to extend or modify the behavior of a class without altering its source code, in accordance to the Open/Closed Principle. They have evaluated the technique by comparing the refactoring opportunities, identified by independent experts, by following the precision and recall approach.

O'Keeffe and Ó Cinnéide [6] proposed a search-based approach for improving the design of object- oriented programs without altering their behavior. They formulated the task of design improvement as a search problem in the space of alternative designs. The quality evaluation functions used to rank the alternative designs were based on metrics from the QMOOD hierarchical design quality model. The refactorings used by the search techniques to move through the space of alternative designs were inheritance-related (Push Down Field/Method, Pull Up Field/Method, Extract/Collapse Hierarchy, Replace Inheritance with Delegation, Replace Delegation with Inheritance and many others). Their approach has been validated by two case studies, in which the results of the employed search techniques (Hill Climbing and Simulated Annealing) and evaluation functions have been compared.

Refactorings, behavior preserving transformations, are claimed to make software easier to understand and to improve software design. Bart Du Bois have validated this quality improvement through his dissertation [7]. He has done it by comparing two reengineering patterns that use refactoring to support program comprehension. The results were studied by formal analysis of the refactoring that improves coupling and cohesion metrics. The results of this research confirm that, indeed, the claimed benefits can occur, and describe how and when the application of refactoring can improve selected quality characteristics.

In the research paper [8], by Gabriele Bavota, Andrea Di Lucia and Rocco Oliveto, propose an Extract Class refactoring method based on graph theory that exploits structural and semantic relationships between methods. The empirical evaluation of the proposed approach highlighted the benefits provided by the combination of semantic and structural measures and the potential usefulness of the proposed method as a feature for software development environments.

The selection of the best classification algorithm for a given dataset is a very widespread problem. It is also a complex one, in the sense it requires to make several important methodological choices. A paper [9] by Vincent Labatut and Hocine Cherifi has focussed on the measure used to assess the classification performance and rank the algorithms.

3. MODEL FOR IDENTIFYING DEGREE OF REFACTORING OPPORTUNITIES

A refactoring opportunity is a set of circumstances that makes it possible to refactor the software, while preserving its original behavior. Refactoring is studied as a process to improve the quality of the code in terms of maintainability, reusability and modifiability. But still, refactoring has no units for measurement. It is difficult to determine whether the process of refactoring has improved or decayed the code. We have thereby proposed a model for refactoring in which we have studied refactoring as a measured phenomenon.

Since, software metrics are useful in determining the quality of the software; we have used some of the object-oriented software metrics based on the principles of abstraction, encapsulation, inheritance and information hiding. The nominal values of metrics are determined and the values of source code metrics are calculated from the UML diagrams of the given code. An ordinal scale of measurement of rank three is proposed.

3.1. Structural Elements of Refactoring Model

3.1.1. Software Metrics

Software metrics are essential to software engineering for measuring software complexity and quality, estimating cost and project effort. The contribution of software metrics towards quality of the software is in general recognized by many software engineering communities [10]. The software metrics [11], [12] used in this research work and their evaluation are defined below:

1) *Number of Methods (NOM):* This is a simple count of the number of operations. Its value should remain between 3 and 7.

2) *Number of Methods Inherited (NMI)*: It is computed using the below given formula:

$$NMI = \frac{NOHO}{HOP} * 100$$

Table 1: Variable Definition for Number of Methods Inherited

The Variable...	Represents the…
NOHO	number of non-redefined inherited operations
HOP	number of inherited operations

3) *Number of Methods Overridden (NMO)*: This is the count of the number of inherited operations that are redefined by the class (Between 0 and 5). A class which inherits services must use them with a minimum of modifications. If this is not the case, the inheritance loses all meaning and becomes a source of confusion.

4) *Depth of Inheritance Tree (DIT)*: This metric provides the position of the class in the inheritance tree. For multiple inheritance, this metric provides the maximum length path. A value of between 0 and 4 respects the compromise between high performance power and complexity introduced due to inheritance.

5) *Number of Children (NOC)*: The number of child classes. It should be limited (Between 1 and 4), notably for reasons of simplicity [13].

6) *Number of Attributes (NOA)*: This is a simple count of the number of attributes, nominally between 2 and 5. A high number of attributes (> 10) probably indicates poor design.

7) *Specialization Index (SIX)*: For a root class, the specialization indicator is zero. For a class, the specialization indicator is obtained through the following equation, where a definition of NMO is Number of Methods Overridden, NMI is Number of Methods Inherited, NMA is Number of Methods Added, and DIT is Depth of Inheritance Tree:

$$SIX = \left(\frac{NMO * DIT}{NMO + NMI + NMA} \right) * 100$$

8) *Coupling between Object Classes (CBO)*: The variable NumberOfLinks represent the number of classes used as associations in all classes. CBO (nominally between 1 and 4) is evaluated as:

$$CBO = \frac{NumberOfLinks}{NumberOfClasses}$$

9) *Abstraction (A)*: The Abstraction metric measures a package's abstraction rate. It is computed as:

$$Abstraction = \frac{Nma}{Nmca} * \frac{Nca}{Nc} * 100$$

The variables Nma, Nmca, Nca and Nc represent the number of abstract operations in all classes, total operations in the abstract classes, number of abstract classes and number of classes(abstract or not) respectively.

10) *Distance from Main Sequence (DMS)*: DMS, the balance between abstraction and instability is obtained through the following expression:

$$DMS = | Abstraction + Instability - 100 |$$

11) *Cyclomatic Complexity (CC)*: It measures the complexity of a module's decision structure. It is denoted as V(G). Nominal range of CC is below 20. It is computed as count of linearly independent paths through a method or methods.

$$V(G) = E - N + P$$

Table 2: Variable Definition for Cyclomatic Complexity

The Variable…	**Represents the…**
E	Number of edges of decision graph
N	Number of nodes of decision graph
P	Number of connected paths

12) *Class Category Relational Cohesion (CCRC)*: This metric measures the rate of cohesion between a package's classes. Its nominal range is between 150% and 350%. CCRC is calculated using the given equation where NumberOfLinks = number of associations and generalizations, and NumberOfClasses = total number of classes:

$$CCRC = \frac{NumberOfLinks}{NumberOfClasses}$$

3.1.2 Ordinal Scale

With ordinal measurement scales there is relative ranking and ordering of an attribute in different categories [14]. The ordinal scale has two properties: Identity (unique meaning) and Magnitude (relationship). It is critical to choose a correct measurement scale to analyze our data, as it should have correct scale/statistic combination. That is, we must use a high powered statistic on a high powered set of data.

3.2. Evaluation of Model

The refactoring model is evaluated with the help of an ordinal scale of rank 3. At each level of an ordinal scale, different set of opportunities are available for refactoring the code. We have determined the relevant opportunities for refactoring at the three levels of ordinal scale:

1. *Low Degree of Refactoring Opportunities*: When the values of software metrics lie within the nominal range of values, we determine it as a good quality code and thus, low refactoring opportunities. On an ordinal scale, it is represented as “Level 0”. According to KentBeck [15], one should let the code do the talking and listen to what it says; one will realize how much refactoring is needed on any given iteration.

2. *Medium Degree of Refactoring Opportunities*: When the values of source-code metrics lie on the edges of nominal range of metrics, we determine it as medium level opportunities for refactoring. On an ordinal scale, it is represented as “Level 1”.

3. *High Degree of Refactoring Opportunities*: When the values of source-code metrics are far from their nominal values, we determine it as a high degree of refactoring opportunities. On an ordinal scale, it is represented as “Level 2”. Refactoring precedes a program modification or extension, bringing the program into a form better suited for the modification [16].

The proposed model is evaluated in the four steps:

Step 1: Identify the projects and draw UML diagrams for each project.
Step 2: Calculate the values of identified object-oriented source-code metrics.
Step 3: Represent each UML on an ordinal scale of measurement.
Step 4: Statistical Analysis of the dataset produced in the previous step.

In this research work, we have applied the proposed model to the three projects, collected from a company Ecologic Corporation. The object-oriented source code metrics, defined in the previous section, are calculated from the UML diagrams drawn for each project. The project details are given the Table 3.

Table 3: Project Details

Project.	Project Name	Number of UML diagrams
1.	ABC CORP	144
2.	Sharma Publishers	109
3.	Friends Sales Corporation	262

Based on the values of source-code metrics, each UML is represented on an ordinal scale for measurement. The dataset, thus produced, is analyzed with the help of a Naïve Bayesian Classifier. The various result metrics are used to determine the performance of the model.

3.3. Performance Analysis of the Refactoring Model

A machine learning tool, weka (Waikato Environment for Knowledge Analysis), is used to carry out the statistical analysis of the dataset of each project. The weka code is written in Java and is supported by Windows, Linux and MAC OS X platforms. The data, in this research, is imported in the form of ARFF file. The Naïve Bayesian Classifier is used for analyzing the data. The naïve Bayesian classifier provides a simple approach, with clear semantics, to represent, use and learn the probabilistic knowledge [17]. The algorithm is designed to minimize processing time and efficiently select the attributes that have the greatest importance; however, we can control the data that is used by the algorithm. The Microsoft Naive Bayes algorithm supports several attributes that affect the behavior, performance, and accuracy of the resulting model. The algorithm calculates the probability of every state of each input column, given each possible state of the predictable column [18]. The outputs are drawn from the confusion matrix obtained by the classifier. A confusion matrix [19] contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. The 3x3 classification problems of the three projects are drawn in the Tables 4, 5, and 6 below:

Table 4: 3x3 Classification of ABC CORP

a	b	c	◄------ Classified as
266	75	58	| a=0
91	147	44	| b=1
65	39	115	| c=2

Table 5: 3x3 Classification of Sharma Publishers

a	b	c	◄------ Classified as
291	69	33	| a=0
79	183	38	| b=1
44	66	97	| c=2

Table 6: 3x3 Classification of Friends Sales Corporation

a	b	c	◄------ Classified as
418	101	24	| a=0
114	295	33	| b=1
45	55	115	| c=2

The transformation of 3x3 classifications of the projects into 2x2 confusion matrix is done as specified in [19]. The Table 7, 8, and 9 shows the classification of the three projects at three levels 0, 1, and 2 separately. The true positives and false positives are obtained from the confusion matrix, which are then used to calculate the result metrics. The result metrics of the different projects are then compared to find out the opportunities for refactoring of the projects.

Table 7: 2x2 Confusion Matrix for ABC CORP

	a	Not a		b	Not b		c	Not c
a	266	133	b	147	135	c	115	102
Not a	156	345	Not b	114	504	Not c	104	579

Table 8: 2x2 Confusion Matrix for Sharma Publishers

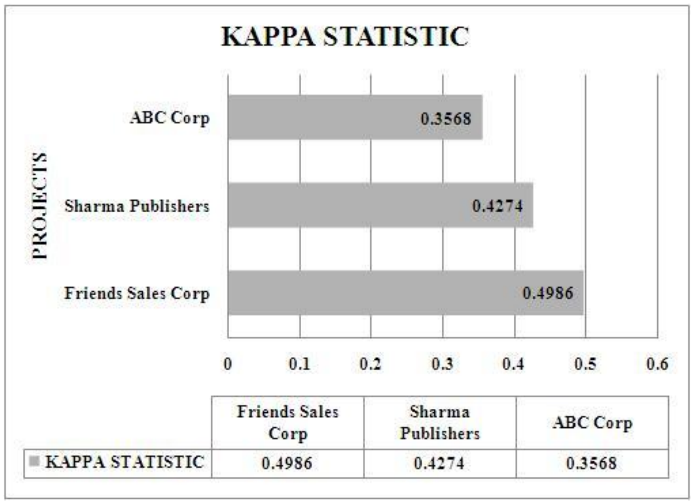
	a	Not a		b	Not b		c	Not c
a	291	102	b	183	117	c	97	71
Not a	123	384	Not b	135	465	Not c	110	622

Table 9: 2x2 Confusion Matrix for Friends Sales Corporation

	a	Not a		b	Not b		c	Not c
a	418	125	b	295	147	c	115	57
Not a	159	498	Not b	156	602	Not c	100	928

3.3.1 Kappa Statistic

Kappa is a measure of agreement with desirable properties. It is used for statistical measurement for qualitative objects. Kappa statistics measure levels of agreement between two observers and make allowance for the degree of agreement that would occur by chance alone [20]. It's calculated by taking the agreement expected by chance away from the observed agreement and dividing by the maximum possible agreement. The comparison of Kappa for three projects is shown in Figure 1.



	Friends Sales Corp	Sharma Publishers	ABC Corp
▨ KAPPA STATISTIC	0.4986	0.4274	0.3568

Figure 1: Comparison based on Kappa Statistic

From Figure 1, it is clear that value of kappa for Project Friends Sales Corp (=0.4986) is maximum, which indicates that this project has a good quality code, thus there are less opportunities for refactoring. Similarly, kappa for project ABC Corp is minimum (= 0.3568) and has the maximum refactoring opportunities to make the code better. While, Sharma Publishers (= 0.4274) has moderate refactoring opportunities.

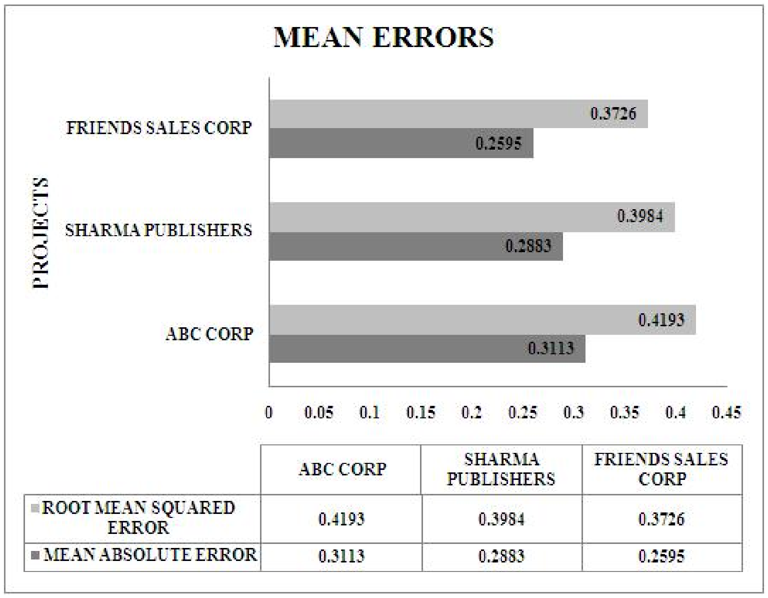
3.3.2 Mean Absolute Error and Root Mean Squared Error

The Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of data, without considering their direction. It measures accuracy for continuous variables. The MAE is a linear score which means that all the individual differences are weighted equally in the average.

$$MeanAbsoluteError = \frac{Sum(AbsoluteErrorPerIns\tan ce)}{NumberOfIns\tan ces}$$

The standard deviation of the errors, also called Root Mean Square Error (RMSE), is the measure of typically spread of data over the regression line. The RMSE is a quadratic scoring rule which measures the average magnitude of the error. The difference between predicted and corresponding observed values are each squared and then averaged over the sample. Finally, the square root of the average is taken.

Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable. For each instance in the test set, weka obtains a distribution. The distribution of mean errors for the three projects is drawn with the help of a bar graph in Figure 2.



	ABC CORP	SHARMA PUBLISHERS	FRIENDS SALES CORP
▨ ROOT MEAN SQUARED ERROR	0.4193	0.3984	0.3726
■ MEAN ABSOLUTE ERROR	0.3113	0.2883	0.2595

Figure 2: Comparison based on Mean Absolute Error and Root Mean Squared Error

From the Figure 2, it can be noted that value of MAE and RMSE are maximum for ABC CORP (thus, it has maximum opportunities for refactoring), while minimum for Friends Sales Corp (thus, it has minimum refactoring opportunities).

3.3.3 F-Measure

The F-Measure is simply a combined measure for precision and recall, given by:

$$F - Measure = \frac{2 * \mathrm{Pr}ecision * \mathrm{Re}call}{\mathrm{Pr}ecision + \mathrm{Re}call}$$

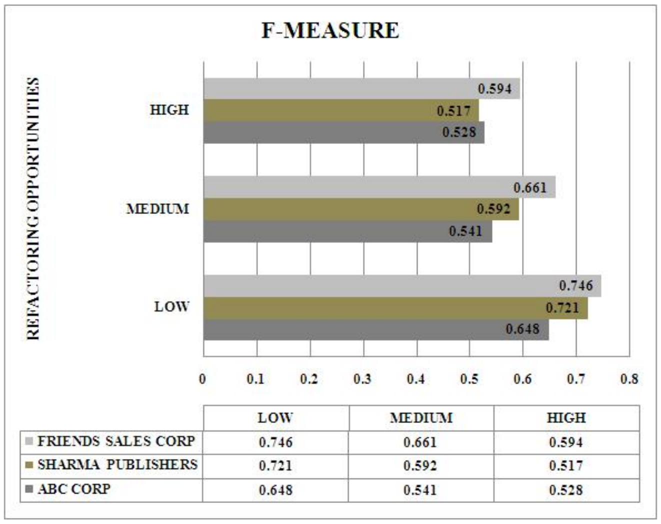
The precision is the fraction of retrieved instances that are relevant, while Recall is the fraction of relevant instances that are retrieved. Both precision and recall are therefore based on an understanding and measure of relevance. The Precision and Recall are calculated using the equations:

$$\mathrm{Pr}ecision = \frac{NumberOfTruePositives}{NumberofTruePositives + FalsePositives}, \mathrm{Re}call = \frac{TruePositives}{TruePositives + FalseNegatives}$$

The F-measure corresponds to the harmonic mean of Predicted Positive Value (PPV) and True Positive Rate (TPR), therefore it is class-specific and symmetric. It is also known as Sørensen’s similarity coefficient, Dice’s coincidence index and Hellden’s mean accuracy index [9]:

$$\mathrm{F1} = \frac{2(PPV1 * TPR1)}{PPV1 + TPR1}$$

$$= \frac{2 * TruePositives}{2TruePositives + FalseNegatives + FalsePositives}$$



	LOW	MEDIUM	HIGH
■ FRIENDS SALES CORP	0.746	0.661	0.594
■ SHARMA PUBLISHERS	0.721	0.592	0.517
■ ABC CORP	0.648	0.541	0.528

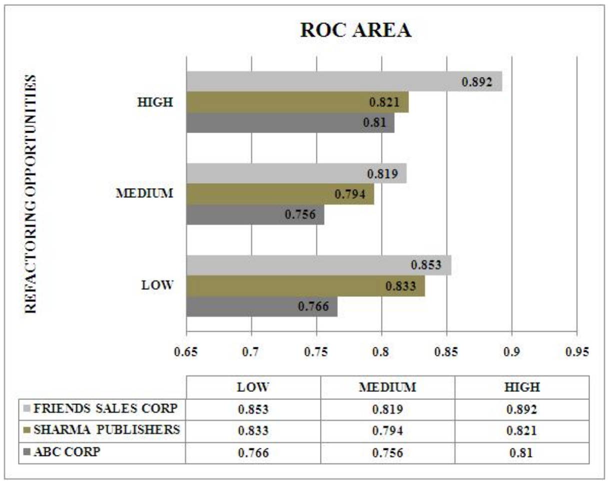
Figure 3: Comparison based on F-Measure

The comparison of three projects, based on the value of f-measure is drawn with the help of a bar graph, as shown in Figure 3. The values of f-measure are calculated at the three levels of refactoring (high, medium, and low). Each project is studied for refactoring at three levels of ordinal scale. A project with highest f-measure has high precision and recall (which means, high accuracy), thus lowest refactoring opportunities.

From the figure 3, it can be noted that at low level of refactoring opportunities; Friends Sales Corp has highest value of f-measure; Sharma Publishers has the moderate value and ABC CORP has lowest value. It can be interpreted that Friends Sales Corp has minimum refactoring opportunities. Same trend is followed at medium level opportunities. While, at high level of refactoring opportunities, where minimum value of f-measure supports high degree of refactoring opportunities, it can be interpreted that Sharma Publishes has high refactoring opportunities.

3.3.4 Area under Receiver Operating Characteristics (AUC)

The two values (TPR= true positive rate and FPR= false positive rate) are better visualized with the help of ROC Curve (Receiver Operating Characteristic). ROC analysis helps to diagnose cost/benefit analysis of decision making. The area under the ROC curve is a widely used measure of performance of supervised classification rules. AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0 [21].



	LOW	MEDIUM	HIGH
▨ FRIENDS SALES CORP	0.853	0.819	0.892
▨ SHARMA PUBLISHERS	0.833	0.794	0.821
▨ ABC CORP	0.766	0.756	0.81

Figure 4: Comparison based on ROC Area

The ROC area of the three projects, at three levels of refactoring opportunities, can be compared with the help of the bar graph drawn in Figure 4. The Project which has highest ROC area is said to have most positively predicted values. Thus, have the minimum refactoring opportunities. It can be interpreted from the Figure 4 that Friends Sales Corp has highest ROC area and ABC CORP has lowest ROC area. The ROC Curves can be drawn for each project (Figure 5). The project with best curve has best predicted values.

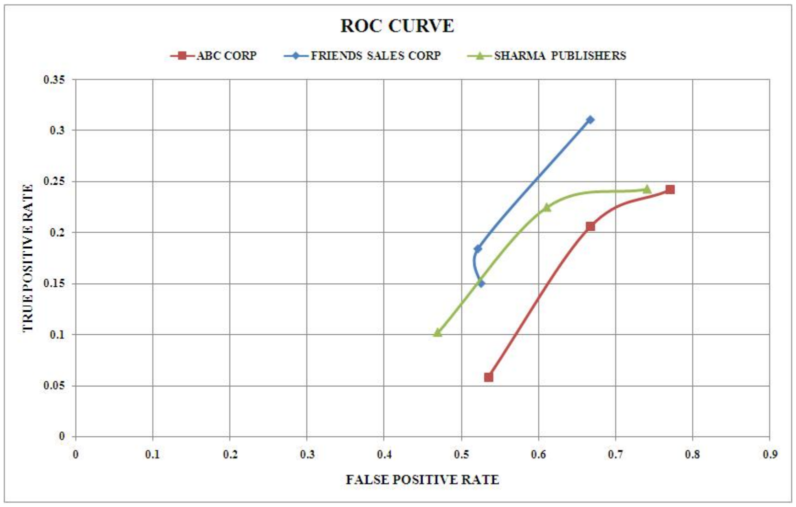


Figure 5: ROC Curve for three Projects

From the Figure 5, it can be interpreted that ROC curve is best for the Friends Sales Corporation, thus the classifier has given the best performance for this project.

4. Conclusions

In this research work, we have studied refactoring as a measured entity. The research is completed in four parts. In the first part, we have collected three projects from a company and UML diagrams are drawn for each project. In the second part, we have identified object-oriented source-code metrics, useful in determining quality of the software. Also, we have drawn an ordinal scale of measurement at three levels of refactoring opportunities namely high, medium and low, represented as 0, 1, and 2 respectively. In the third part, we have represented each UML, based on the values of source-code metrics, on an ordinal scale. The dataset thus produced is analyzed using a Naive Bayes machine algorithm in the last part of research. A machine learning tool, weka, is used to analyze the data. The results (Kappa Statistic, Mean Absolute Error, Root Mean Squared Error, F-Measure, ROC Curve, and Area under ROC) are compared for each project to identify the project that has maximum and minimum refactoring opportunities.

ACKNOWLEDGEMENTS

I would like to thank my research guide Mrs. Sonka Jindal, Assistant Professor at Shaheed Bhagat Singh State technical Campus, Ferozepur, for her utmost guidance in my research work. Also, I would thank Mr. Puneet Arora, Head of Ecologic Corp, Panchkula for providing projects used in the research. At last, I would thank my parents for their superlative support.

REFERENCES

[1] Drozdz, M., Kourie, D. G., Watson, B. W., & Boake, A. (2006). “Refactoring tools and complementary techniques”, AICCSA, Vol. 6, pp. 685-688
[2] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999) *Refactoring: Improving the Design of Existing Code,* Addison Wesley.
[3] Tsantalis, N., & Chatzigeorgiou, A. (2010) “Identification of refactoring opportunities introducing polymorphism”, Journal of Systems and Software, Vol. 83, No. 3, pp. 391-404.
[4] Murphy-Hill, E., Parnin, C. and Black, A. P. (2011) “How we refactor, and how we know it” IEEE Transactions on Software Engineering, 2011
[5] Ge, X., DuBose, Q. L., & Murphy-Hill, E. (June 2012). “Reconciling manual and automatic refactoring”, In Software Engineering (ICSE), 2012 34th International Conference, IEEE, pp. 211-221
[6] O’Keeffe, M., Ó Cinnéide, M. (2008) “Search-based refactoring for software maintenance”, The Journal of Systems and Software. Vol. 81, No. 4, pp. 502–516.
[7] Du Bois, B. (2006) “A study of quality improvements by refactoring”, Dissertation Abstracts International, Vol. 68, No. 2
[8] Bavota, G., Lucia, A. De and Oliveto, R (2011) "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," Journal of Systems and Software, Vol. 84, No. 3, pp. 397-414.
[9] Labatut, V. & Cherifi, H. (2012) “Accuracy Measures for the Comparison of Classifiers.” arXiv preprint arXiv: 1207.3790.
[10] Pressman, R. (1992). *Software Engineering: a Practitioner's Approach*, 3rd edition, McGraw-Hill.
[11] http://support.objecteering.com/objecteering6.1/help/us/metrics/metrics_in_detail
[12] http://www.imagix.com/links/software-metrics.html.
[13] Chidamber , S. R. & Kemerer, C. F. (1994), “A metrics suite for object oriented design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493.
[14] http://stattrek.com/statistics/measurement-scales.aspx
[15] Beck, K. (2003) *Test driven development: By example*, Addison-Wesley Professional.
[16] Thompson, S. (2005) “Refactoring functional programs”, In Advanced Functional Programming, Springer Berlin Heidelberg, pp. 331-357.
[17] John, George H. and Langley, Pat (1995) “Estimating Continuous Distributions in Bayesian Classifiers”, In Proceedings: Eleventh Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Mateo, pp. 338-345.
[18] Heckerman, D., Geiger, D. & Chickering, D.M. (1995) “Learning Bayesian networks: The combination of knowledge and statistical data”, Machine learning, Vol. 20, No. 3, pp. 197-243.
[19] Kohavi, R. & Provost, F. (1998) “Glossary of terms”, Machine Learning, Vol. 30, No. 2/3, pp. 271-274.
[20] Cross, S. S. (1996) “Kappa statistics as indicators of quality assurance in histopathology and cytopathology”, Journal of clinical pathology, Vol. 49, No. 7, pp. 597-599.
[21] Fawcett, T. (2006) “An introduction to ROC analysis”, Pattern recognition letters, Vol. 27, No. 8, pp. 861-874.