International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 15 (2018) pp. 11816-11823

Empirical Evaluation of Software Design Patterns using Classification Algorithms based Design Metrics

Moha Gupta
M. Tech
Central University of Punjab, India.

Satwinder Singh
Assistant Professor
Central University of Punjab, India.

Abstract

Software engineering has become very important for the technology. It includes the study and application of the engineering in designing, developing and maintaining the software. The aim of software engineering is to produce quality software to be delivered on time keeping it within budget which satisfies the requirement. The patterns can be detected by code inspections, automatic techniques, dynamic techniques such as testing and assertions, using machine learning and by static techniques. These design patterns can be detected from the design metrics using machine learning algorithms This study emphasis on detecting software design pattern-based design metrics using machine learning algorithm. The results surely depict that the design metrics plays an important role in predicting the software design patterns. The algorithms used are Naive Bayes and SVM and they were able to predict the patterns from the software metrics and gave good perceptiveness.

Keywords: Naïve Bayes, Design Patterns, SVM, Machine learning, Software engineering, Coupling

INTRODUCTION

In context to software engineering, the software quality computes that how the software will be designed and also how the software conforms for that design. The software quality has been upgraded from past few three decades. The reason possible behind this is that the companies have started using new technologies and techniques in the software developmental process like CASE tools or object-oriented models etc. All the aspects of the software quality are interconnected and can have impact on others as well, the impact can be negative or positive. The main reason for involving software quality in software development is to get the final product built as per the requirements specified. As in today's world, software engineering has become very important for the technology. It includes the study and application of the engineering in designing, developing and maintaining the software. The aim of software engineering is to produce quality software to be delivered on time keeping it within budget which satisfies the requirement. It has a wide range of applications in every field like in business software, used for management and controlling of financial activities, in artificial intelligence it is used for problem- solving techniques, in web-based software used as an interface.

Software pattern detection is an important part in the software development process. Pattern detection can be made by various methods using different types of algorithms. There are many studies that can be done using the software metrics. One of them includes the detection of software patterns. Patterns can be detected using manual and automatic approaches where automatic method include Machine Learning algorithms. The software patterns can also be detected using software metrics. Supervised learning is being implemented for detecting of software patterns from the software design metrics. The software design metrics can be found by using some software which will then be treated as an input to the algorithm for detecting the software patterns. At the end, the design pattern prediction model will be proposed from the software metrics. Some studies are done on the design patterns and design metrics.

LITERATURE REVIEW

(Chidamber et. al. 1994), developed six design metrics namely WMC, DIT, CBO, LCOM, NOC and RFC. These design metrics were evaluated analytically against another author Weyukar, who proposed a set of the measurement principles. Also a tool was developed for automatic data collection for collecting data of these metrics at the two field sites for examining the feasibility. The C&K metrics have some limitations that it doesn't account for the complexity which arises from polymorphism and encapsulation, which was further studied by (Li et. al. 1993), in which Li studied two more commercial systems. Out of total six metrics by C&K, five of them namely LCOM, RFC, DIT, WMC and NOC helped in predicting the maintenance effort. Still many researchers kept on proposing modifications on the metrics like (Basili et. al. 1996), studied student projects and found that WMC was correlated with the defects while LCOM was not. Also, Basili studied that CBO, RFC, DIT and NOC were also correlated. (Uchiyama et. al. 2011) proposed a pattern detection technique using the software metrics and some machine learning methods. In their technique, it judges candidate for the role which compose the design patterns by using the machine learning and measurement of metrics. It distinguishes the design patterns where the class structure are similar and also suppresses the false negatives. It was proved that this technique is better than the old techniques. (Malhotra et. al. 2012) discussed the relationship between fault proneness and object oriented metrics using logistic regression algorithm. Performance of the predicted models is evaluated using Receiver Operating Characteristic (ROC) Analysis. In other discussion, study used a statistical model which was derived from logistic regression to calculate the threshold values of Kemerer, object oriented and Chidamber metrics.

Threshold effects at the various risk levels was calculated using the threshold values and he has also validated the use of these threshold values on public domain, dataset, KC1 from NASA and 2 open source promise datasets i.e. JEdit and IVY using various data mining classifiers and machine learning algorithms. Three different open source datasets i.e. Tomcat, Sakura and Ant were used to perform inter project validation. And the results depicted that proposed threshold methodology performs well on the project having same or similar characteristics.

(Okutan et. al. 2012) used Bayesian networks for determining the probabilistic influential relationships between defect proneness and software metrics. Other two metrics i.e. source code quality (LOCQ) and number of developers (NOD) were also used with the metrics already present in Promise Data Repository. In total of 9 data sets of open source Promise Data Repository were used. Using these metrics and datasets it was found that response for class (RFC), lack of coding quality (LOCQ) and lines of code (LOC) were the most effective metrics while weighted method per class (WMC), lack of cohesion methods (LCOM) and coupling objects (CBO) were less effective metrics on the defect proneness. (Patil et. al. 2013), discussed the performance based on the incorrect and correct instances of the data classification using J48 and Naïve Bayes algorithms. The algorithms are used to compare and evaluate the bank dataset and to maximize the true positive rates while minimizing the false positive rates using the WEKA tool. After experimentation it was concluded that Naïve Bayes generated 184 correct instances while J48 generated 203 correct instances. Random Forest is widely used in various fields and have various applications, it is a group of classification algorithms, it is especially used with larger datasets due to some of its best features such as OOB error detection, Variable Importance measure and Proximity (Zakariah, 2014). The paper also discusses many applications of the Random Forest for classification of the dataset such as Gene Classification, Network intrusion detection, Email spam detection, Credit Card fraud detection and Text classification. This paper focuses on use of the Random Forest features in the applications like for selection of the required gene among the genes, variable importance measure can be used which helps in removing the less significant gene, and for detection of the intrusion the outlier property of Random Forest is used.

In another paper**,** (Yadav et. al. 2015) integrated 3 classifiers with simple k-means clustering algorithm while the integration was applied on the bug data set. It was concluded that k-means & Bayes net gives 0.0012 RMSE and 0.0002 MAE error and also it took less time to build. At the end out of k-means & J48, k-means & Decision tree and k-means & Bayes net, it was founded that k-means & Bayes net is the best algorithm integration. (Arnu et. al. 2016), a meta-analysis was conducted of 2001-2015 papers found where the novel Random Forest algorithm was proposed and is then compared to already proposed Random Forest algorithm. Many limitations were studied like based on performance measures, the estimation way of these measures and the methodology for comparing the algorithms. And it was found that almost one-third of results from the Random Forest papers, improvement was not found in the performance when comparisons were drawn using statistical tests.

Classifier models work by the learning patterns between corresponding binary label and attributes of the software which indicates whether a defect exists or not (Perreault et. al., 2017). In the study, the performance of five different classifiers were evaluated like Support Vector Machines (SVM), Naive Bayes, K-Nearest Neighbours, Neural Networks and logistic regression. Also, both F1-score and accuracy performance were measured and ANOVA was used to test the significance. Each and every classifier was made to run on five different datasets from NASA's repository data program. It was discovered that all the five models were able to detect software defects by using software features having comparatively high degree of certainty, but, models Support Vector Machines (SVM) and Naïve Bayes outperformed from the remaining models for some of the datasets. In another paper, (Gupta et. al., 2018) discussed the prediction of design patterns based software metrics using the supervised machine learning algorithms i.e. Random Forest and J48. The study used four eclipse versions Eclipse 3.2, Eclipse 3.3, Eclipse 3.6 and Eclipse 3.7 for the software metrics and software patterns. The design patterns were predicted from the metrics and was discovered that the algorithms gave very good perceptiveness in the prediction of software patterns.

CHIDAMBER AND KEMERER METRICS

Software metrics are being used by the engineers to evaluate and perform necessary features in a software project. The software metrics control and identify the important parameters which are affecting software development. Also, it helps in evaluating design patterns via better platform. There are a number of metrics used and proposed but the supreme software metrics are used that are easy to learn, understand, are efficient and effective. Chidamber and Kemerer metrics are used which are specially designed keeping in mind the object oriented code. The main objective of the Chidamber & Kemerer metrics is to measure different object-oriented metrics like inheritance, complexity and coupling. Chidamber and Kemerer introduced first coupling metrics for the object-oriented system which is known as CBO (Coupling Between Objects), defined as a metric with number of non-inherited related couples with the other classes. Different inheritance metrics are also introduced like Depth of Inheritance Tree (DIT), Number of Children (NOC) etc. The importance of inheritance is proved which says that reduction in the amount of the software maintenance is very necessary. Also the reuse of inheritance is proved to be more reliable, understandable and maintainable (Chidamber & Kemerer, 1994).

Originally six metrics were proposed by Chidamber and Kemerer which are discussed below:

1. **CBO (Coupling Between Objects)-** It is defined as the count of number of the classes which are coupled with a certain class or method of a class accessing the variable or object or calls the method of other classes. Such calls had to be counted in one and the other directions, which says that CBO of class X

includes the size of set of classes that references class X and those which class X references. As it is a set therefore every class is being counted once even the reference performs in both directions i.e. X references Y and Y References X, then Y is counted only once.

2. **DIT (Depth of Inheritance Tree)-** It is the count of number of classes which a specific class inherits from. Some consequences by C & K are suggested based on DIT as it makes more difficult to predict the behaviour of class as, the deeper the class hierarchy, it will inherit more number of methods. Also as large number of methods and classes are involved, therefore deeper trees comprise significant design complexity.
3. **LCOM (Lack of Cohesion of Methods)-** C & K defined it as number of the pairs of methods which share references to the instance variables**.** Each and every pair of methods combination was assessed in the class, and if the pair of method share references to any of the instance variable then count is reduced by 1 and if the pair don't share references then it is escalated by 1. As low the value of LCOM the better the cohesion of the class.
4. **NOC (Number of Children)-** It is defined as the number of contiguous subclass of a class by C & K. According to C & K as the inheritance is a type of reuse, so more the number of children, then more the level of reuse. Also more the number of children, more the chance of improper abstraction of parent class. Also it might be possible for misuse of subclassing.
5. **RFC (Response for Class)-** It is defined as the size of response set of the class, and the response set is defined as the set of methods which can be executed in reply to the message obtained by object of the class. As it's a set therefore only once each method called is counted, it doesn't matter the number of times it is called.
6. **WMC (Weighted Methods for Class)-** It is proposed as the sum of all complexities of methods in a class. Each method is assigned a value of one as a complexity making WMC as equal to number of the methods in that class, rather than using cyclomatic complexity. The view of C & K for WMC was that the complexity of the methods and the number of methods involved tells the amount of effort and time required for developing and maintaining the class.

SOFTWARE DESIGN PATTERNS

In software engineering, design patterns are a repeatable solution for some commonly occurring problem in the software design. It is basically a template or a description used for solving problems in different situations. There are 23 design patterns which can be categorized in 3 categories. These design patterns were developed by the Gang of Four (GoF). The Gang of Four are authors of the book 'Design Patterns: Elements of Reusable Object-Oriented Software'. These authors are namely, Erich Gamma, John Vlissides, Richard Helm and Ralph Johnson. The 3 design patterns are:

1. **CREATIONAL PATTERNS-** These types of pattern provide way for creating an object while hiding the logic of creation. These patterns provide an interface for the creation of families of dependent or related objects without even specifying the concrete class. This type of pattern gives the program more flexibility for determining which object has to be created for the given use case. These types of patterns are all about the class instantiation. They can be further categorized into object-creational patterns and class-creational patterns. The object-creational patterns effectively use delegation to get work done while the class-creation patterns effectively use inheritance in instantiation process.
2. **STRUCTURAL DESIGN PATTERNS-** This type of pattern is concerned with the composition of classes and objects. In structural pattern, the interface of a class is converted to another interface which the client is expecting. Also concept of interface is being used to define ways for composing objects to obtain the new functionalities. This type of pattern uses inheritance for composing interfaces.
3. **BEHAVIORAL DESIGN PATTERNS-** These types of patterns are mainly concerned with the communication between the objects. This pattern acts as an interpreter i.e. when a language is given, the representation of grammar is defined with an interpreter which interprets the sentences in the language by representation. It also defines common functionality for group of classes.

EVALUATION AND DISCUSSIONS

The work is performed in various phases having different objectives. The phases are described in detail as follows-

Phase 1: Software Metrics Data Collection

The metrics as explained is a quantitative measure of the software characteristics and a very important depicter of the software. They provide a very efficient and effective ground for selecting the most appropriate and best way to save software development money, time and effort. The metrics used for analysis are Eclipse 3.2, 3.3 were analysed by the Understand® software. Though many metrics were yielded but only the object oriented metrics were selected finally. The metrics selected were LCOM, DIT, WMC, NOC and RFC. Once the metrics were analysed then the design patterns were examined. The descriptive statistics of the versions of Eclipse are shown in table 1-4.

Table 1: Eclipse 3.2

Metrics	Mean	Standard Deviation	MIN	MAX	Percentile		
					25%	50%	75%
LCOM	31.519	42.528	0	518	11	40	68
DIT	1.9342	3.5626	0	30	5	7	12
CBO	11.126	15.012	0	266	3	6	13
NOC	0.5298	8.8074	0	964	0	0	0
RFC	42.176	93.992	0	2741	13	19	43
WMC	7.3454	19.67	0	981	1	3	8

Table 2: Eclipse 3.3

Metrics	Mean	Standard Deviation	MIN	MAX	Percentile		
					25%	50%	75%
LCOM	30.4783	36.8532	0	100	4	12	68
DIT	0.9156	1.4531	0	10	0	0	1
CBO	10.83	14.164	0	224	3	6	13
NOC	0.5152	8.9687	0	1053	0	0	0
RFC	41.527	68.237	0	2383	13	19	41
WMC	7.1282	15.278	0	1100	1	3	8

From the tables it was observed that the value of NOC was 0 for all the classes and the value of DIT metric was 1 for all the classes that means only 1 level of inheritance is there.

Phase 2: Software Patterns Data Collection

The software design patterns are the repeatable solutions in the software design. They help in implementing efficient, effective and standardized solutions to the software design. They are also known as tested solutions for the object-oriented problems. The patterns of the versions of Eclipse 3.2 and Eclipse 3.3 were analysed by Web of Patterns (WOP) tool. The patterns extracted are among those four types of design patterns as discussed above as Creational, Structural and Behavioral.

Table 3: Design Pattern Data Information

Source Code	Pattern Participant	Number of pattern
Eclipse 3.2	Singleton	5
	Proxy	3
Eclipse 3.3	Adapter	51
	Template	61
	Singleton	29
	Proxy	12

Phase 3: Experimentation Setup:

The experiments were performed by selecting a platform for conducting machine learning algorithms. Weka was used as the platform for the experiment. The machine learning algorithms used were Naïve Bayes and SVM. The models for detection of software patterns were created and examined for the efficiency and accuracy in Weka through 10-fold cross validation. Cross validation technique is used for evaluating the predictive models by partitioning the main sample to training set for training the model and testing to evaluate the sample. In 10-fold cross validation the sample is randomly being partitioned into 10 equal size subsamples. From these 10 subsamples, 9 subsamples are sued for training the model while the rest 1 is used for testing the model. Then this 10-fold validation technique is repeated 10 times having each 10 subsamples being used exactly once. After the 10 results, the average is taken to get a single estimation.

Different performance measures were obtained for each run of the model. Each metrics prediction model was used for detecting design patterns in the succeeding versions of Eclipse. The tests were performed using Naïve Bayes and SVM with the default settings in Weka.

To detect the software design patterns-based design metrics, the design metrics should be as accurate as possible. The resultant design metrics prediction model has following measures:

Phase 4: Observations Recorded:

To detect the software design patterns-based design metrics, the design metrics should be as accurate as possible. The resultant design metrics prediction model has following measures:

From Table No. 4, it can be inferred that the design metrics prediction models created are accurate for detecting the presence of software design patterns as well as the absence of software design patterns. As seen from the table, the values of precision and recall are good enough which implies that the models are successful for detection of software design patterns. This implies that the models used are able to detect the design patterns accurately.

Naïve Bayes Based Design Metrics Prediction Model:

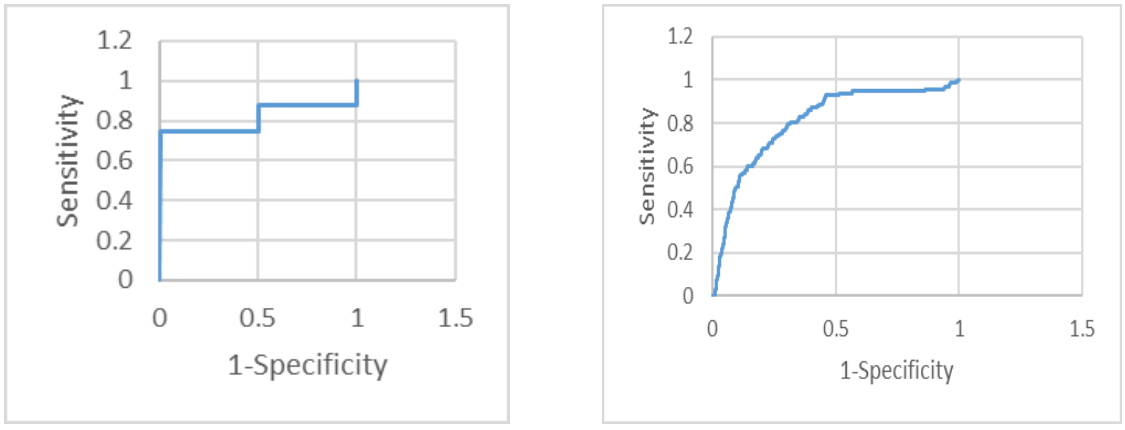
The design metrics prediction models based on Naïve Bayes have the measures as under:

Table 4: Performance measures of the design metrics prediction models

Source Code	Algorithm	Precision	Recall	F-measure	ROC
Eclipse 3.2	Naïve Bayes	0.853	0.889	0.894	0.812
Eclipse 3.3		0.877	0.892	0.902	0.806
Eclipse 3.2	SVM	0.665	0.678	0.684	0.625
Eclipse 3.3		0.689	0.695	0.702	0.640

Table 5: Performance measures of Naïve Bayes based Design Metrics Prediction model

Algorithm	Prediction Model	Applied On	Precision	Recall	F-measure	ROC
Naïve Bayes	Eclipse 3.2	Eclipse 3.2	0.875	0.897	0.899	0.872
		Eclipse 3.3	0.688	0.694	0.691	0.498
	Eclipse 3.3	Eclipse 3.3	0.991	0.994	0.991	0.906



(a) **(b)**

Figure 1: ROC curves on application of Naïve Bayes based metrics prediction model for detection of software design pattern in subsequent version. (a) Eclipse 3.2 design metric prediction model. (b) Eclipse 3.3 design metric prediction model.

The value of ROC is an indication of performance of the classifier on various versions of Eclipse. The ROC for Eclipse 3.2 comes out to be 81.2% which means very good perceptiveness, Eclipse 3.3 ROC value is 80.6% which also

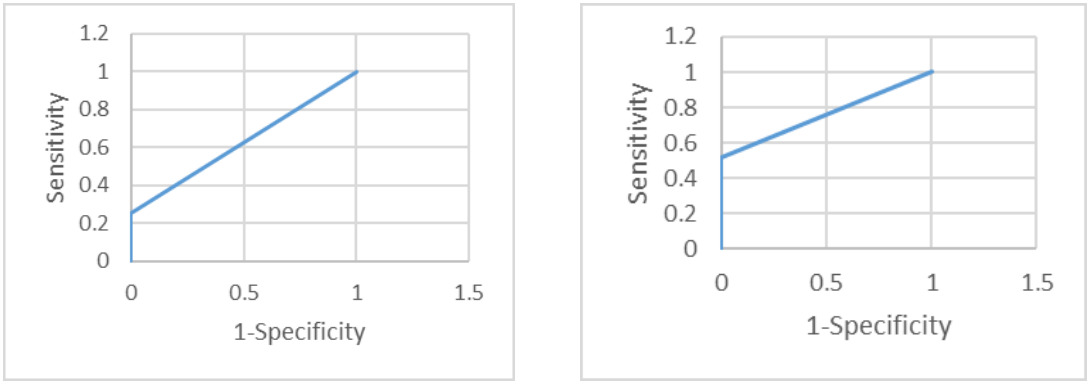
means very good perceptiveness which implies good perceptiveness, therefore, it shows that it is able to detect patterns based design metrics.

SVM Based Design Metrics Prediction Model

The model SVM was also used for detection of software design patterns based design metrics same as of the Naïve Bayes model. The performance measures of SVM is shown in the given Table:

Table 6: Performance measures of SVM based Design Metrics Prediction model

Algorithm	Prediction Model	Applied On	Precision	Recall	F-measure	ROC
SVM	Eclipse 3.2	Eclipse 3.2	0.875	0.881	0.889	0.856
		Eclipse 3.3	0.688	0.694	0.691	0.500
	Eclipse 3.3	Eclipse 3.3	0.975	0.981	0.989	0.760



(a) **(b)**

Figure 2: ROC curves on application of SVM based metrics prediction model for detection of software design pattern in subsequent version. (a) Eclipse 3.2 design metric prediction model. (b) Eclipse 3.3 design metric prediction model.

The value of ROC is an indication of performance of the classifier on various versions of Eclipse. The ROC for Eclipse 3.2 comes out to be 62.5% which means average perceptiveness, Eclipse 3.3 ROC value is 64.0% which also means average perceptiveness, which implies that it gives average perceptiveness and therefore, it shows that it is able to detect patterns based design metrics.

CONCLUSION

In this paper, various Chidamber and Kemerer metrics were discussed which are very useful in software engineering. the 3 types of software patterns were also discussed i.e. creational, structural and behavioral. some tools were already been used for the analysis of software patterns like JHotDraw 7 and JDeodrant on different versions of the software. The results surely depict that the design metrics plays an important role in predicting the software design patterns. The algorithms used are Naive Bayes and SVM and they were able to predict the patterns from the software metrics and the algorithm Naïve Bayes gave very good perceptiveness while the algorithm SVM gave an average of perceptiveness. The study used the two types of classification algorithms but for future work the various other regression algorithm or the other unsupervised algorithms can also be used for the prediction of software design patterns. also, some other tools can also be used for analysing the software metrics and software patterns.

REFERENCES

[1]. Ackerman. L and Gonzalez. C, (2011) The value of pattern implementations', The World of Software Development Journal, Computer Science. Vol.32 Issue. 6, pp. 28-32.

[2]. Al-Ja'afer, J., & Sabri, K. (2004). Chidamber-Kemerer (CK) and Lorenze-Kidd (LK) metrics to assess Java programs. King Abdullah II school for information technology, University of Jordan, Jordan.

[3]. Anvik, J. (2007). Assisting bug report triage through recommendation (Doctoral dissertation, University of British Columbia).

[4]. Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10), 751-761.

[5]. Bhargava, N., Sharma, G., Bhargava, R., & Mathuria, M. (2013). Decision tree analysis on j48 algorithm for data mining. Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering, 3(6).

[6]. Challagulla, V. U. B., Bastani, F. B., Yen, I. L., & Paul, R. A. (2008). Empirical assessment of machine learning based software defect prediction techniques. International Journal on Artificial Intelligence Tools, 17(02), 389-400.

[7]. Chang, C. P., Chu, C. P., & Yeh, Y. F. (2009). Integrating in-process software defect prediction with association mining to discover defect pattern. Information and software technology, 51(2), 375-384.

[8]. Chaurasia, V., & Pal, S. (2014). Data mining approach to detect heart diseases.

[9]. Cheikhi, L., Al-Qutaish, R. E., Idri, A., & Sellami, A. (2014). Chidamber and kemerer object-oriented measures: Analysis of their design from the metrology perspective. International Journal of Software Engineering & Its Applications, 8(2), 359-374.

[10]. Chidamber, S. R., Darcy, D. P., & Kemerer, C. F. (1998). Managerial use of metrics for object-oriented software: An exploratory analysis. IEEE Transactions on software Engineering, 24(8), 629-639.

[11]. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493.

[12]. Cline, M. P. (1996). The pros and cons of adopting and applying design patterns in the real world. Communications of the ACM, 39(10), 47-49.

[13]. D'Ambros, M., Lanza, M., & Robbes, R. (2010, May). An extensive comparison of bug prediction approaches. In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on (pp. 31-41). IEEE.

[14]. D'Ambros, M., Lanza, M., & Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering, 17(4-5), 531-577.

[15]. Dickinson, W., Leon, D., & Podgurski, A. (2001, July). Finding failures by cluster analysis of execution profiles. In Proceedings of the 23rd international conference on Software engineering (pp. 339-348). IEEE Computer Society.

[16]. Fowler. M, 'Patterns' (2003), IEEE software, 20(2).Retrieved May 8, 2011 from Fowler On Patterns.

[17]. Gamma, E. (1995). Design patterns: elements of reusable object-oriented software. Pearson Education India.

[18]. Goel, B. M., & Bhatia, P. K. (2012). Analysis of reusability of object-oriented system using CK metrics. International Journal of Computer Applications, 60(10).

[19]. Hampherey Watts S.(1995) A discipline for software Engineering reading, Ma,Addison Wesley.

[20]. Hegedűs, P., Bán, D., Ferenc, R., & Gyimóthy, T. (2012). Myth or reality? analyzing the effect of design patterns on software maintainability. In Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity (pp. 138-145). Springer, Berlin, Heidelberg.

[21]. Jiang, S., & Mu, H. (2011, July). Design patterns in object oriented analysis and design. In Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on (pp. 326-329). IEEE.

[22]. Kalmegh, S. R. (2015). Comparative analysis of weka data mining algorithm randomforest, randomtree and ladtree for classification of indigenous news data. International Journal of Emerging Technology and Advanced Engineering, 5(1), 507-517.

[23]. Lovedeep and Varinder Kaur Arti (2014) Application of Data mining techniques in software engineering International journal of electrical, electronics and computer system(IJEECS) Volume-2 issue-5, 6.

[24]. Malhotra, R., & Jain, A. (2012). Fault prediction using statistical and machine learning methods for improving software quality. Journal of Information Processing Systems, 8(2), 241-262.

[25]. Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), 154-181.

[26]. Olague, H. M., Etzkorn, L. H., Gholston, S., & Quattlebaum, S. (2007). Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on software Engineering, 33(6), 402-419.

[27]. Pal, A. K., & Pal, S. (2013). Analysis and mining of educational data for predicting the performance of students. International Journal of Electronics Communication and Computer Engineering, 4(5), 1560-1565.

[28]. Perreault, L., Berardinelli, S., Izurieta, C., & Sheppard, J. (2017). Using Classifiers for Software Defect Detection. In 26th International Conference on Software Engineering and Data Engineering, SEDE (pp. 2-4).

[29]. Qureshi, M., & Qureshi, W. (2012). Evaluation of the design metric to reduce the number of defects in software development. arXiv preprint arXiv:1204.4909.

[30]. SAHIB. (2014). Final Report On Defect Prediction Model of Static Code Features for Cross-Company and Cross-Project Software.

[31]. Seliya, N., & Khoshgoftaar, T. M. (2007). Software quality analysis of unlabeled program modules with semisupervised clustering. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 37(2), 201-211.

[32]. Suresh, Y., Pati, J., & Rath, S. K. (2012). Effectiveness of software metrics for object-oriented system. Procedia Technology, 6, 420-427.

[33]. Uchiyama, S., Washizaki, H., Fukazawa, Y., & Kubo, A. (2011, March). Design pattern detection using software metrics and machine learning. In First International Workshop on Model-Driven Software Migration (MDSM 2011) (p. 38).

[34]. Yu, P., Systa, T., & Muller, H. (2002). Predicting fault-proneness using OO metrics. An industrial case study. In Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on (pp. 99-107). IEEE.

[35]. Zakariah, M. (2014). Classification of large datasets using Random Forest Algorithm in various applications: Survey. money, 4(3), 189-198.