Implication of Data Mining and Machine Learning in Software Engineering Domain for Software Model, Quality and Defect Prediction

Anurag Sinha[1] **Shubham singh**[2] **Devansh Kashyap**[3]

Department of computer science and IT, Student, Amity University Jharkhand Ranchi, Jharkhand(India), 834001**anuragsinha257@gmail.com**[1]

Department of computer science and Engineering, Galgotias University, Greater Noida, U.P, 203201
shubham932singh@gmail.com[2]

Department of computer science and IT, Kalinga Institute of Industrial Technology, Bhubaneswar , Odisha, 751024
devanshkahyap23@gmail.com[3]

Abstract: Software metrics have a direct link with measurement in software engineering. Correct measurement is the prior condition in any engineering fields, and software engineering is not an exception, as the size and complexity of software increases, manual inspect becomes a harder task. Most Software Engineers worry about the quality of software, how to measure and enhance its quality. The overall objective of this study was to asses and analysis's software metrics used to measure the software product. Developers have attempted to improve software quality by mining and analyzing software data. In any phase of software development life cycle (SDLC), while huge amount of data is produced, some design, security, or software problems may analyze software data helps to handle these problems and lead to more accurate and timely delivery of software projects. Various data mining and machine learning studies have been conducted to deal with software engineering tasks such as defect prediction, effort estimation, etc. This study shows the open issues and presents related solutions and recommendations in software engineering, applying data mining and machine learning techniques. Software quality is a field of study and practice that describes the desirable attributes of software products. The performance must be perfect without any defects. Software quality metrics are a subset of software metrics that f software defect prediction model helps in early detection of defects and contributes to their efficient removal and producing a quality software system based on several metrics. The main paper is to help developers identify defects based on existing software metrics using data mining techniques and thereby improve the software quality. In this paper, various classification are revisited which are employed for software defect prediction using software metrics in the literature.

Keywords: Software Metrics, Software Quality software engineering tasks, data mining, text mining, classification, clustering Software Defect Prediction, Software Metrics, Classification.

I. Introduction

In context of software engineering, software quality refers to software functional quality and software structural quality. Software functional quality reflects functional requirements whereas structural quality highlights non- functional requirements. Software metrics focus on the quality aspect of the product, process and project. In this paper the main emphasis is on software product. The objective of software product quality engineering is to achieve the required quality of the product through the definition of quality requirements and their implementation, measurement of appropriate quality attributes and evaluation of the resulting quality. Software quality measurement [15] is about quantifying to what extent a system or software possesses desirable characteristics namely Reliability, Efficiency, Security, Maintainability and (adequate) Size. This can be performed through qualitative or quantitative means or a mix of both. In both cases, for each desirable characteristic, there are a set of measurable attributes like Application Architecture Standards, Coding Practices, Complexity, Documentation, Portability and Technical & Functional volumes. The existence of these attributes in a piece of software or system tends to be correlated and associated with this characteristic. Software Defect Prediction [SDP] plays an important role in the active research areas of software engineering. A software defect is an error, bug, flaw, fault, malfunction or mistake in software that causes it to create a wrong or unexpected outcome. The major risk factors related with a software defect which is not detected during the early phase of software development are time, quality, cost, effort and wastage of resources. Defects may occur in any phase of software development. Booming software companies focus concentration on software quality, particularly during the early phase of the software development .Thus the key objective of any organization is to determine and correct the defects in an early phase of Software Development Life Cycle [SDLC]. To improve the quality of software, data mining techniques have been applied to build predictions regarding the failure of software components by exploiting past data of software components and their defects. This paper reviewed the state of art in the field of software defect management and prediction, and offered data mining techniques in brief mutation. Software metrics provide quantitative means to control the software development and the quality of software products. They are necessary to identify where the resources are needed, and are a crucial source of information for decision-making. A large number of measures have been proposed in the literature. A particular emphasis has been given to the measurement of design artifacts, in order to help assess quality early on during the development process. Assessment of design quality is objective, and the measurement can be automated. But how do we know what measures actually capture important quality aspects? The ISO/IEC standard (14598) states that internal metrics are especially helpful when they are related to external quality attributes, e.g., maintainability, reusability, etc. Many different approaches have been proposed to build empirical assessment models; for example, they can be mathematical models (case of statistical techniques like linear and logistic regression) or artificial intelligence-based models (case of machine-learning techniques). Our work deals then with the construction of efficient and/or intelligible assessment models for quality factors, especially, maintainability and reusability. To build these models, we have used different Machine-Learning (ML) algorithms. We are interesting in this study, to compute their performance and to estimate their intelligibility, on software engineering data. Performance refers to a quantitative dimension, generally expressed by the accuracy of the model, whereas intelligibility evokes the explicitness and the understandability of the model. Thus, we first present in section 2 the different ML algorithms we have used. In section 3, we describe the empirical process we follow, then present and discuss the models produced, from the performance and intelligibility perspectives. Finally, in section 4, conclusions are outlined. In recent years, researchers in the software engineering (SE) field have turned their interest to data mining (DM) and machine learning (ML)-based studies since collected SE data can be helpful in obtaining new and significant information. Software engineering presents many subjects for research, and data mining can give further insight to support decision-making related to these subjects. Figure 1 shows the intersection of three main areas: data mining, software engineering, and statistics/math. A large amount of data is collected from organizations during software development and maintenance activities,suchas requirement specifications, design diagrams, source codes, bug reports, program versions, and so on. Disciplines such as DM, SE, and statistics. This study presents a comprehensive literature review of existing research and offers an overview of how to approach SE problems using different mining techniques. Up to now, review studies either introduce SE data descriptions [1], explain tools and techniques mostly used by researchers for SE data analysis [2], discuss the role of engineers [3], or focus only on a specific problem in SE such as defect prediction [4], design pattern [5], or effort estimation [6].

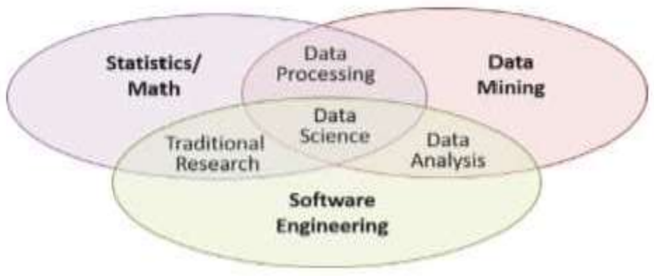


Figure 1- The intersection of data mining and software engineering with other areas of the field

2. MACHINE-LEARNING FOR BUILDING MODELS

ML is a well-established field of Artificial Intelligence (AI) with many accomplishments (conferences, journals, techniques, tools,). Many approaches have been developed, and most of the work done in ML has focused on s algorithms. Starting from the description of classified examples, these algorithms produce definitions for each class. The choice of which specific learning algorithm we should use is a critical step. A large number of approaches have been developed and supervised learning is certainly the most prolific one. The main strength of approaches like decision trees, rules, Bayesian Networks (BNs), and to a lesser degree, Case Based Learning (CBL), is that they produce models we can incorporate in a decision process, where, knowledge-based system architecture keeps a good separation between what we consider as an explicit expert knowledge (the produced models, e.g., rules, trees, BNsand the procedures that exploit this knowledge. On the other hand, approaches like Artificial Neural Networks (ANN) and Support Vector Machines (SVM) are considered as “black box” ones; a part from defining initial parameters (network. architecture, random numbers, …), we have no other role than to feed them with i watch them train and await the output. Thus, the multilayer perception is probably the most widely used ANN architecture to solve classification problems with supervised learning. The standard training procedure for the multilayer perception uses the back-propagation (BP) algorithm [1] or one of its derivatives. There are many solutions proposed by many researchers to overcome the slow converge rate problem. Resilient Back Propagation (RBP) is one of them [2]. Finally, SVM is a relatively new learns introduced by Vapnik in 1995 for solving two pattern recognition problems [3]. SVM has proven to be effective for many classification tasks such as text classification, facial expression recognition, gene analysis, and many others. We mu that no single approach/technique can uniformly outperform other approaches over all data must find a compromise between the characteristics we want to emphasis. The next section will present models obtained using these ML approaches, on data related to some software product quality attributes. Their performance and intelligibility will be discussed.

3. LITERATURE SURVEY

Peng He et al. conducted an empirical study on software defect prediction with a simplified metric set [2]. Research has been conducted on 34 releases of 10 open source projects available at PROMISE repository. The finding indicates the result of top-k metrics or minimum metric subset provides acceptable output compared with benchmark predictors. The simplified or mini mertic set works well in case of minimum resources. Grishma BR et al. investigated root cause for fault prediction by applying clustering techniques and identifies the defects occurs in various phases of SDLC. In this research they used COQUALMO prediction model to predict the fault in a software and applied various clustering algorithms like k-means, agglomerative clustering, COBWEB, density based scan, expectation maximization and farthest first. Implementation was done using Weka tool. Finally they c that k-means algorithm works better when compared with other algorithms [1]. Anuradha Chug et al. used three supervised [classification] learning algorithms and three unsupervised [clustering] learning algorithms for predicting defects in software. NASA MDP datasets were run by using Weka tool. Several measures like recall and f-measure are used to evaluate the performance of both classification and clustering algorithms. By analyzing different classification algorithms Random Forest has the highest accuracy of MC1 dataset and also yields highest value in recall, fmeasure and receiver operating characteristic [ROC] curve and it indicates minimum number of root mean square errors in all circumstances. In an unsupervised algorithm k-means has the lowest number of incorrect clustered instances and it takes minimum time for predicting faults [3]. Jaechang Name et al. applied Hetrogeneous Defect Prediction [HDP] to predict defects in with-in and across projects with different datasets. Metric selection, metrics matching and building a prediction model are the 3 methods used in this work. In this research they used various datasets from NASA, PROMISE, AEEEM, MORPH and SOFTLAB. Source and target datasets are used with different metric sets. For selecting metrics feature selection techniques such as gain ratio, chisquare, relief-F and significance attribute selection are applied to the source. To match source and target metrics various analyzers like Percentile based Matching (PAnalyzer), Kolmogorov – Smirnov test based matchiong (KSAnalyzer), Spearman's Correlation based Matching (SCOAnalyzer) are used. Cutoff threshold value is applied to all pair scores and poorly matched metrics are removed by comparison. Area Under the Receiver Operator Characteristic Curve [AUC] measure is used to compare the performance between different models. HDP is compared with 3 baselines – WPDP, CPDP-CM, CPDP-IFS by applying win/loss/tie evaluation. The experiments are repeated for 1000 times and Wilcoxon signed rank test ($P<0.05$) is applied for all AUC values and baselines. Performance is measured by counting the total number of win/loss/tie. When a cutoff threshold value gets increased in PAnalyzer and KSAnalyzer, the results (win) also gets increased. Logistic Regression (LR) model works better when there is a linear relationship between a predictor and bug-prone [4]. Logan Perreault et al. applied classification algorithm such as naïve bayes, neural networks, support vector machine, linear regression, K-nearest neighbor to detect and predict defects. The authors used NASA and tera PROMISE datasets. To measure the performance they used accuracy and f1 measure with clearly well defined metrics such as McCabe Metrics and Halstead Metrics. 10-fold cross validation is used in which 90% of data are used for training and 10% of data are used for testing. ANOVA and tukey test was done for 5 dataset and 5 response variables. 0.05 is set as significance level for PC1, PC2, PC4 and PC5 dataset and 0.1 as PC3 dataset. Weka tool is used for implementation. Implementations of these 5 algorithms are available on Github repository. Finally the authors conclude that all datasets are similar and they are written in C or C++ and in future the work can be extended by selecting the datasets that are written in Java and instead of using weka tool for implementation some other tool can also be used [5]. Ebubeogu et al. employed predictor variables like defect density, defect velocity and defect introduction time which are derived from defect acceleration and used to predict the total number of defects in a software. MAChine – Learning – Inspired [MACLI] approach is used for predicting defects. The proposed framework for defect prediction has two phases. 1) Data preprocessing phase. 2) Data analysis phase [6].

4. Software Defect Prediction

A software defect is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most defects arise from mistakes and errors made by people in either a program's source code or its design, or in frameworks and operating systems used by such programs, and a few are caused by compilers producing incorrect code. Software Defect Prediction Model refers to those models that try to predict potential software defects from test data. There exists a correlation between the software metrics and the fault proneness of the software. A Software defect prediction models consists of independent variables (Software metrics) collected and measured during software development life cycle and dependent variable (faulty or non faulty). There are different data mining techniques for defect prediction. Data mining is the analysis step of the "Knowledge Discovery in Databases" process, or KDD, a process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further analysis. Data Mining can be divided into two tasks: Predictive tasks and descriptive tasks. Predictive task is to predict the value of a specific attribute (target/dependent variable)based on the value of other attributes (explanatory). Descriptive task is to derive patterns (correlation, trends, and trajectories) that summarize the underlying relationship between data. There are various data mining techniques used for software defect predictions which are discussed below.

1. Regression: It is a statistical process to evaluate the relationship among variables. It analyses the relationship between the dependent or response variable and independent or predictor variables. The relationship is expressed in the form of an equation that predicts the response variable as a linear function of predictor variable. [42, 24, 51, 25] Linear Regression: $Y=a+bX+u$

2. Association Rule Mining: It is a method for discovering interesting relationships between variables in large databases. It is about finding association or correlations among sets of items or objects in database. It basically deals with finding rules that will predict the occurrence of item based on the occurrence of other items. [11, 17, 40,26]

3. Clustering: Clustering is a way to categorize a collection of items into groups or clusters whose members are similar in some way. It is task of grouping a set of items in such a way that items in the same cluster are similar to each other and dissimilar to those in other clusters. [27, 34, 17, 30]

4. Classification: It consists of predicting a certain outcome based on a given input. Classification technique use input data, also called training set where all objects are already tagged with known class labels. The objective of classification algorithm is to analyze and learns from the training data set and develop a model. This model is then used to classify test data for which the class labels are not known. [43, 27, 30,6, 22]. The various classification techniques are given below.

a. Neural Networks: Neural Networks are the non linear predictive models which can learn through training and resemble biological neural networks in structure. A neural network consists of interconnected processing elements called neurons that work together in parallel within a network to produce output. [22, 21, 47, 42]

b. Decision Trees: A decision tree is a predictive model which can be used to represent both classification and regression models in the form a tree structure. It refers to a hierarchical model of decisions and their consequences. It is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf nodes represent a classification or decision. [39, 31, 37]

c. Naive Bayes: It is based on Bayes theorem with independence assumption between predictors. Naive Bayes Classifier is based on the assumption that the presence or absence of a particular feature of a class in not related to the presence or absence of any other feature. [21, 14, 28]

d. Support Vector Machines: SVM are based on the concept of decision planes that define decision boundaries. A decision plane is the one that separates between a set of objects having different class membership. SVM is primarily a classifier method that performs classification task by constructing hyper plane in a multidimensional space that separates cases of different class labels. It supports both regression and classification. [50, 10, 29] e. Case Based Reasoning: Case based reasoning means solving new problems based on the similar past problems and using old cases to explain new situations. It works by comparing new unclassified records with known examples and patterns. A simple example of a case based learning algorithm is k-nearest neighbor algorithm. It is simple algorithm that stores all available cases and classifies new cases based on a similarity measure i.e. distance function. [39]

Approach of Software Defect Prediction.

Mostly three approaches are performed to evaluate prediction models.
3.1 With-in Project Defect Prediction [WPDP]
3.2 Cross Project Defect Prediction [CPDP] for Similar Dataset
3.3 Cross Project Defect Prediction [CPDP] for Heterogeneous Dataset

4.1With-in Project Defect Prediction a prediction model can be constructed by collecting histori data from a software project and predicts faults in the same project are known as WPDP. WPDP performed best, if there is enough quantity of historical data available to train models. Turhan, Burak, et al. [15] suggested that software defect prediction areas typically focus on developing defect prediction models with existing local data (i.e. within project defect prediction). To apply these models, a company should have a data warehouse, where project metrics and fault related information from past projects are stored. Zimmermann et al. [11] notify that defect prediction performs better within projects as long as there is an adequate data to train models. That is, to construct defect predictors, we need access to historical data. If the data is absent, Company Defect Prediction (CCDP) can be applied. The drawbacks of with-in project defect prediction are: It is not constantly possible for all projects to collect such historical data 100% accuracy cannot be achieved using WPDP. On the other hand, historical data is often not presented for new projects and for many organizations. In this case, successful defect prediction is complicated to accomplish. To tackle this problem, cross project defect prediction strategy was applied.

4.2 Cross Project Defect Prediction [CPDP] for Similar Dataset CPDP is used in a mode such that a project does not have sufficient historical data to train a model. So that, a prediction model is developed for one project and it has been applied for some other project or across project. i.e. transferring prediction models from one project to another project [10]. The drawbacks of applying CPDP is that it desires projects that have similar metric set, implication that the metric sets must be equal among projects. As an outcome, present techniques for CPDP are complicated to relate across projects with dissimilar dataset.

4.3 Cross Project Defect Prediction [CPDP] for Heterogeneous Dataset To deal with the inadequacy of using only similar dataset for CPDP, in Project Defect Prediction A prediction model can be constructed by collecting historical data from a software project and predicts faults in the same project are known as WPDP. WPDP performed best, if there is enough quantity of historical data available to train models. Turhan, Burak, et al. [15] suggested that software defect areas typically focus on developing defect prediction models with existing local data (i.e. within project defect prediction). To apply these models, a company should have a data warehouse, where project metrics and fault related acts are stored. Zimmermann et al. [11] notify that defect prediction performs better within projects as long as there is an adequate data to train models. That is, to construct defect predictors, we need access to historical data. If the data is absent, Cross Company Defect Prediction (CCDP) can be in project defect It is not constantly possible for all projects to collect such historical data Hence 100% accuracy cannot be achieved using WPDP. er hand, historical data is often not presented for new projects and for many organizations. In this case, successful defect prediction is complicated to accomplish. To tackle this problem, cross project defect prediction

4.4Cross Project Defect Prediction [CPDP] for Hetrogeneous Dataset To deal with the inadequacy of using only similar dataset for CPDP, heterogeneous defect prediction [HDP] technique was proposed to predict defects across projects with imbalanced metric sets [4].

Software Defect Prediction Techniques

To improve the effectiveness and quality of software development and to predict defects in software, various data mining techniques can be applied to different Software Engineering areas. The broadly used SDP techniques are datamining techniques and machine learning techniques are depicted in Figure 2

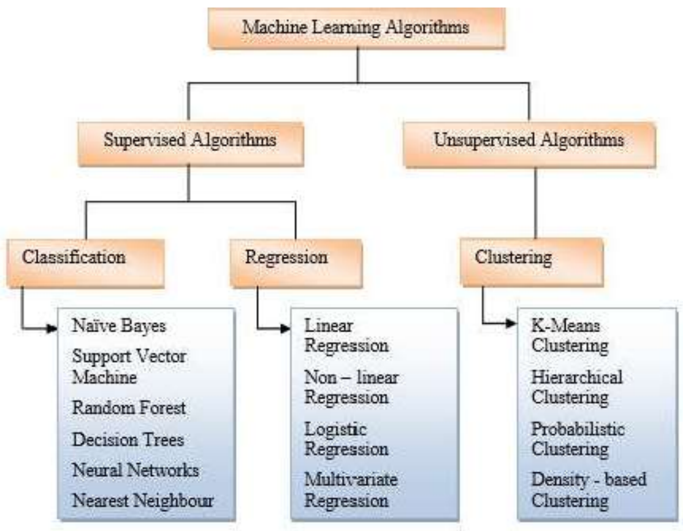


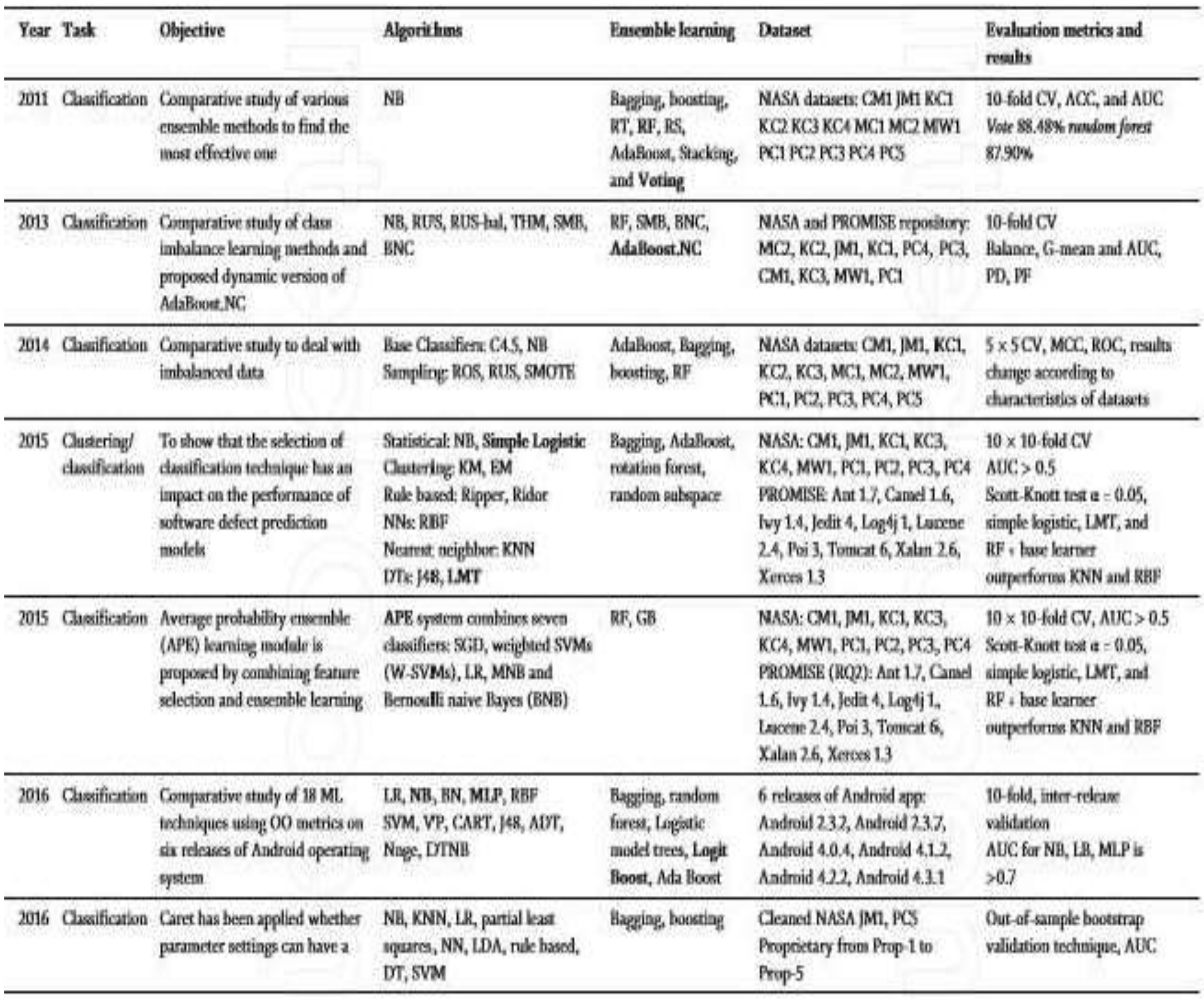
Figure 2: Machine learning algorithms

5. Data mining in defect prediction

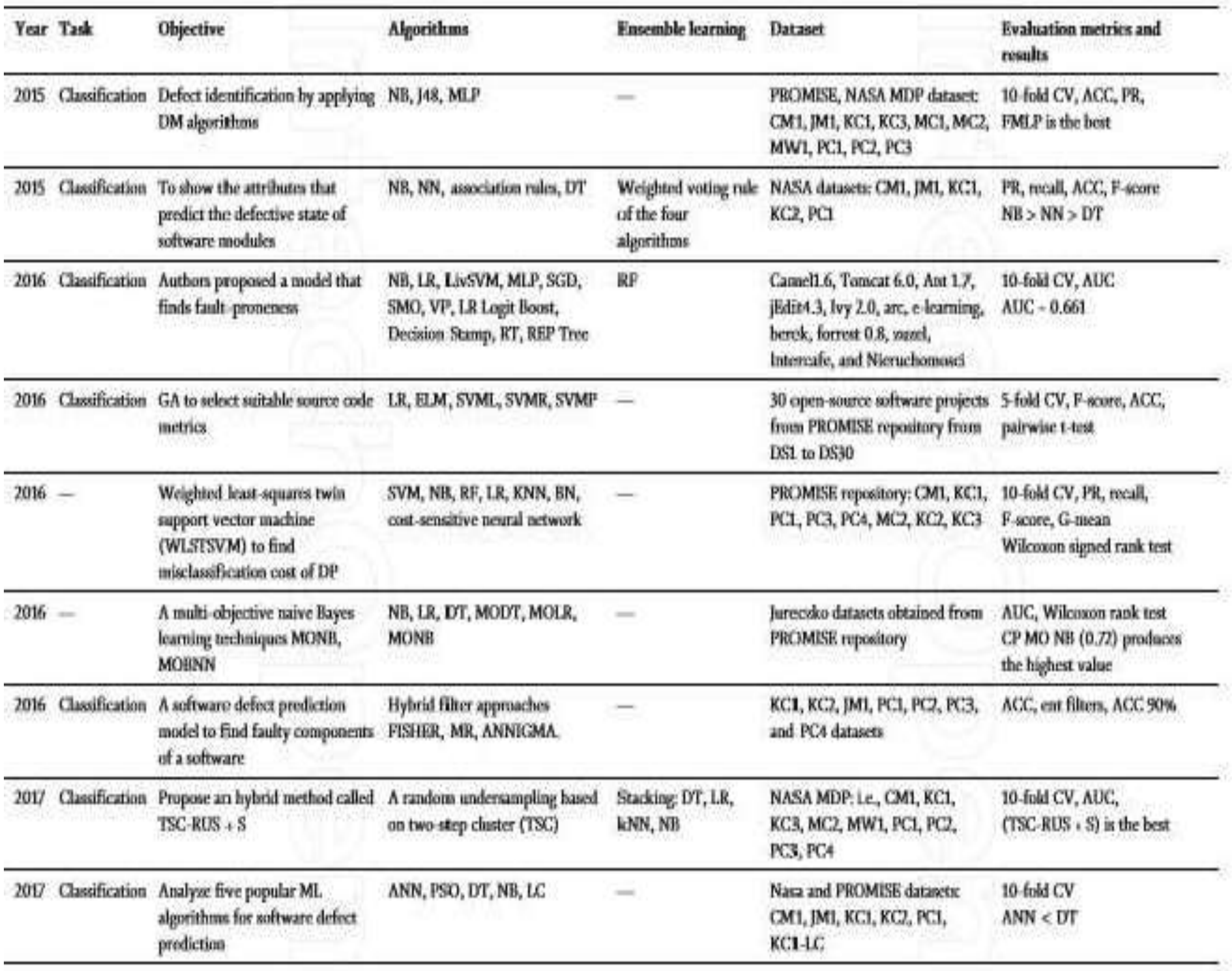
A defect means an error, failure, flaw, or bug that causes incorrect or unexpected results in a system [8]. A software system is expected to be without any defects since software quality represents a capacity of the defect of the product [9]. However, software projects often do not have enough time or people working on them to extract errors before a product is released. In such a situation, defect prediction methods can help to detect and remove defects in the initial stages of the SDLC and to improve the quality of the software product goal of defect prediction is to produce robust and effective software systems. Software systems. Hence, software defect prediction (SDP) is an important topic for software engineering because early prediction of software defects could help to reduce development c more stable software systems. Various studies have been conducted on defect prediction using different metrics such as code complexity. Metrics, object-oriented metrics, and process metrics to construct prediction models [10, 11]. These models can be considered on a cross-project or within-project basis. In within-project defect prediction (WPDP), a model is constructed and applied on the same project [12]. For within project strategy, a large amount of historical defect data is needed. Hence, in new projects that do not have enough data to train, cross-project strategy may be preferred [13]. Cross-project defect prediction (CPDP) is a method that involves applying a prediction model from one project to another, meaning that models are prepared by utilizing historical data from other projects [14, 15]. Studies in the field of CPDP have increased in recent years [10, 16]. However, there are some deficiencies in comparisons of prior studies since they cannot be replicated because of the difference in utilizing evaluation metrics or preparation way of training data. Therefore, Herbold et al. [16] tried to replicate different CPDP methods previously proposed and find which approach performed best in terms of metrics such as F-score, area under the curve (AUC), and Matthews's correlation coefficient (MCC). Results showed that 7- or 8-year approaches may perform better. Another study [17] replicated prior work to demonstrate whether the determination of classification techniques is important. Both noisy and cleaned datasets were used, and the same results were obtained from the two datasets. However, new dataset gave better results for some classification algorithms. For this reason, authors claimed that the selection of classification techniques affects the performance of the model. Numerous defect prediction studies have been conducted using DM techniques. In the following subsections, we will explain these studies in terms of whether they apply ensemble learning or not. Some defect prediction studies in SE are compared.

Defect prediction using ensemble learning techniques Ensemble learning combines several base learning models to obtain better performance than individual models.
These base learners can be acquired with:
i. Different learning algorithms
ii. Different parameters of the same algorithm
iii. Different training sets
The commonly used ensemble techniques bagging, boosting, and stacking are shown in Figure 3 and briefly explained in this part. Bagging (which stands for bootstrap aggregating) is a kind of parallel ensemble. In this method, each model is built independently, and multiple training datasets are generated from the original dataset through random selection of different feature subsets; thus, it aims to decrease variance. It combines the outputs of each ensemble member by a voting mechanism. Boosting can be described as sequential ensemble. First, the same weights are assigned to data instances; after training, the weight of wrong predictions is increased, and this process is repeated as the ensemble size. Finally, it uses a weighted voting scheme, and in this way, it aims to decrease bias. Stacking is a technique that uses predictions from multiple models via a metaclassifier. Some software defect prediction studies have compared ensemble techniques to determine the best performing one [10, 18, 21, 39, 40]. In a study conducted by Wang et al. [18], different ensemble techniques such as bagging, boosting, random tree, random forest,

random subspace, stacking, and voting were compared to each other and a single classifier (NB). According to the results, voting and random forest clearly exhibited better performance than others. In a different study [39].



Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
2011	Classification	Comparative study of various ensemble methods to find the most effective one	NB	Bagging, boosting, RT, RF, RS, AdaBoost, Stacking, and **Voting**	NASA datasets: CM1 JM1 KC1 KC2 KC3 KC4 MC1 MC2 MW1 PC1 PC2 PC3 PC4 PC5	10-fold CV, ACC, and AUC *Vote 88.48% random forest* 87.90%
2013	Classification	Comparative study of class imbalance learning methods and proposed dynamic version of AdaBoost.NC	NB, RUS, RUS-bal, THM, SMB, BNC	RF, SMB, BNC, **AdaBoost.NC**	NASA and PROMISE repository: MC2, KC2, JM1, KC1, PC4, PC3, CM1, KC3, MW1, PC1	10-fold CV Balance, G-mean and AUC, PD, PF
2014	Classification	Comparative study to deal with imbalanced data	Base Classifiers: C4.5, NB Sampling: ROS, RUS, SMOTE	AdaBoost, Bagging, boosting, RF	NASA datasets: CM1, JM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5	5 × 5 CV, MCC, ROC, results change according to characteristics of datasets
2015	Clustering/ classification	To show that the selection of classification technique has an impact on the performance of software defect prediction models	Statistical: NB, **Simple Logistic** Clustering: KM, EM Rule based: Ripper, Ridor NNs: RBF Nearest neighbor: KNN DTs: J48, **LMT**	Bagging, AdaBoost, rotation forest, random subspace	NASA: CM1, JM1, KC1, KC3, KC4, MW1, PC1, PC2, PC3, PC4 PROMISE: Ant 1.7, Camel 1.6, Ivy 1.4, Jedit 4, Log4j 1, Lucene 2.4, Poi 3, Tomcat 6, Xalan 2.6, Xeroes 1.3	10 × 10-fold CV AUC > 0.5 Scott-Knott test α = 0.05, simple logistic, LMT, and RF + base learner outperforms KNN and RBF
2015	Classification	Average probability ensemble (APE) learning module is proposed by combining feature selection and ensemble learning	**APE** system combines seven classifiers: SGD, weighted SVMs (W-SVMs), LR, MNB and Bernoulli naive Bayes (BNB)	RF, GB	NASA: CM1, JM1, KC1, KC3, KC4, MW1, PC1, PC2, PC3, PC4 PROMISE (RQ2): Ant 1.7, Camel 1.6, Ivy 1.4, Jedit 4, Log4j 1, Lucene 2.4, Poi 3, Tomcat 6, Xalan 2.6, Xeroes 1.3	10 × 10-fold CV, AUC > 0.5 Scott-Knott test α = 0.05, simple logistic, LMT, and RF + base learner outperforms KNN and RBF
2016	Classification	Comparative study of 18 ML techniques using OO metrics on six releases of Android operating system	LR, NB, BN, MLP, RBF SVM, VP, CART, J48, ADT, Nnge, DTNB	Bagging, random forest, Logistic model trees, **Logit Boost**, Ada Boost	6 releases of Android app: Android 2.3.2, Android 2.3.7, Android 4.0.4, Android 4.1.2, Android 4.2.2, Android 4.3.1	10-fold, inter-release validation AUC for NB, LB, MLP is >0.7
2016	Classification	Caret has been applied whether parameter settings can have a	NB, KNN, LR, partial least squares, NN, LDA, rule based, DT, SVM	Bagging, boosting	Cleaned NASA JM1, PC5 Proprietary from Prop-1 to Prop-5	Out-of-sample bootstrap validation technique, AUC



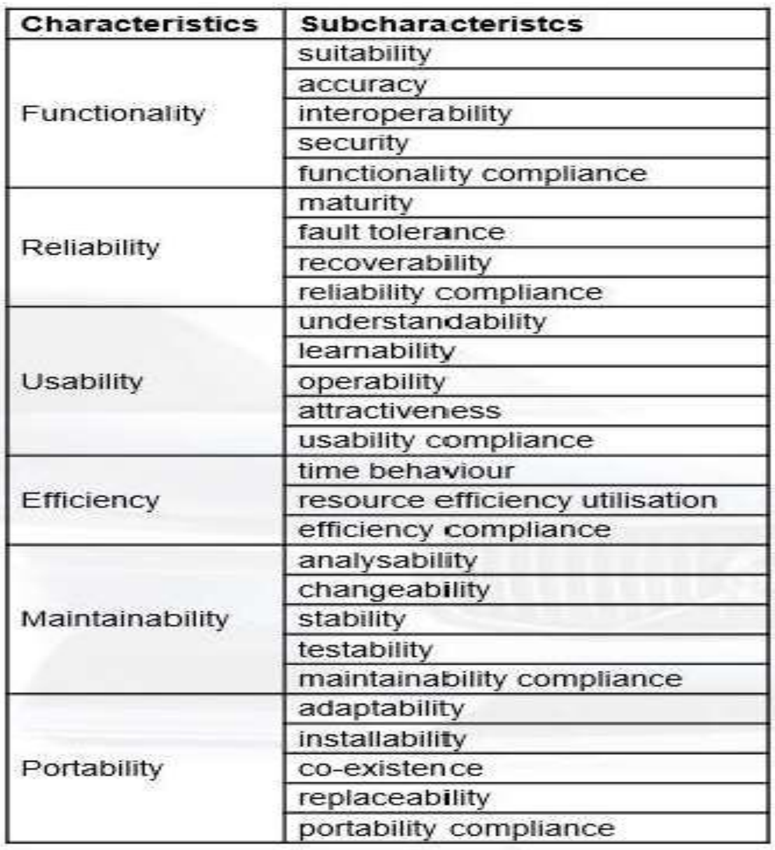
Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
2015	Classification	Defect identification by applying DM algorithms	NB, J48, MLP	—	PROMISE, NASA MDP dataset: CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3	10-fold CV, ACC, PR, FMLP is the best
2015	Classification	To show the attributes that predict the defective state of software modules	NB, NN, association rules, DT	Weighted voting rule of the four algorithms	NASA datasets: CM1, JM1, KC1, KC2, PC1	PR, recall, ACC, F-score NB > NN > DT
2016	Classification	Authors proposed a model that finds fault-proneness	NB, LR, LivSVM, MLP, SGD, SMO, VP, LR Logit Boost, Decision Stamp, RT, REP Tree	RF	Camel1.6, Tomcat 6.0, Ant 1.7, jEdit4.3, Ivy 2.0, arc, e-learning, berek, forrest 0.8, zuzel, Intercafe, and Nieruchomosci	10-fold CV, AUC AUC - 0.661
2016	Classification	GA to select suitable source code metrics	LR, ELM, SVML, SVMR, SVMP	—	30 open-source software projects from PROMISE repository from DS1 to DS30	5-fold CV, F-score, ACC, pairwise t-test
2016	—	Weighted least-squares twin support vector machine (WLSTSVM) to find misclassification cost of DP	SVM, NB, RF, LR, KNN, BN, cost-sensitive neural network	—	PROMISE repository: CM1, KC1, PC1, PC3, PC4, MC2, KC2, KC3	10-fold CV, PR, recall, F-score, G-mean Wilcoxon signed rank test
2016	—	A multi-objective naive Bayes learning techniques MONB, MOBNN	NB, LR, DT, MODT, MOLR, MONB	—	Jureczko datasets obtained from PROMISE repository	AUC, Wilcoxon rank test CP MO NB (0.72) produces the highest value
2016	Classification	A software defect prediction model to Find faulty components of a software	Hybrid filter approaches FISHER, MR, ANNIGMA.	—	KC1, KC2, JM1, PC1, PC2, PC3, and PC4 datasets	ACC, ent filters, ACC 90%
2017	Classification	Propose an hybrid method called TSC-RUS + S	A random undersampling based on two-step cluster (TSC)	Stacking: DT, LR, kNN, NB	NASA MDP: i.e., CM1, KC1, KC3, MC2, MW1, PC1, PC2, PC3, PC4	10-fold CV, AUC, (TSC-RUS + S) is the best
2017	Classification	Analyze five popular ML algorithms for software defect prediction	ANN, PSO, DT, NB, LC	—	Nasa and PROMISE datasets: CM1, JM1, KC1, KC2, PC1, KC1-LC	10-fold CV ANN < DT

6. Machine Learning Approach for Quality Assessment and Prediction in Large Software Organizations

ISO 9126 defines quality as "The totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs" While ISO 25000 takes the following approach to quality: "capability of software product to satisfy stated and implied needs when used under specified conditions" • Assessing software quality early in the development process are essential to identify and allocate resources where they are needed most • Software metrics provide quantitative means to control software product and quality, • Software quality estimation models establishes relationships between desired software quality characteristics and measurable attributes, • These models can be based on statistical techniques such as regression models or logical models, • Since logical models such as those based on decision trees or rule sets are white-box models their interpretation and thus also preferred.

Software Quality • Software metrics have long been used for monitoring and controlling software process, asses and/or improve software quality, • Metrics collection and analysis is part of daily work activities in large software development organizations, • Mature software development organizations also widely use the information model of ISO/IEC standard 15939 as means of identifying the information needs and implementing measurement systems.
In this paper we propose how ML based approaches can be used within the ISO/IEC 15939 information model framework for effective assessment and prediction of software quality. Framework that uses machine learning approaches within ISO/IEC 15939 information model will enhance the adoption of these techniques in large scale software organizations already using the standard for their information needs.

TABLE 1 SOFTWARE QUALITY



Characteristics	Subcharacteristcs
Functionality	suitability
	accuracy
	interoperability
	security
	functionality compliance
Reliability	maturity
	fault tolerance
	recoverability
	reliability compliance
Usability	understandability
	learnability
	operability
	attractiveness
	usability compliance
Efficiency	time behaviour
	resource efficiency utilisation
	efficiency compliance
Maintainability	analysability
	changeability
	stability
	testability
	maintainability compliance
Portability	adaptability
	installability
	co-existence
	replaceability
	portability compliance

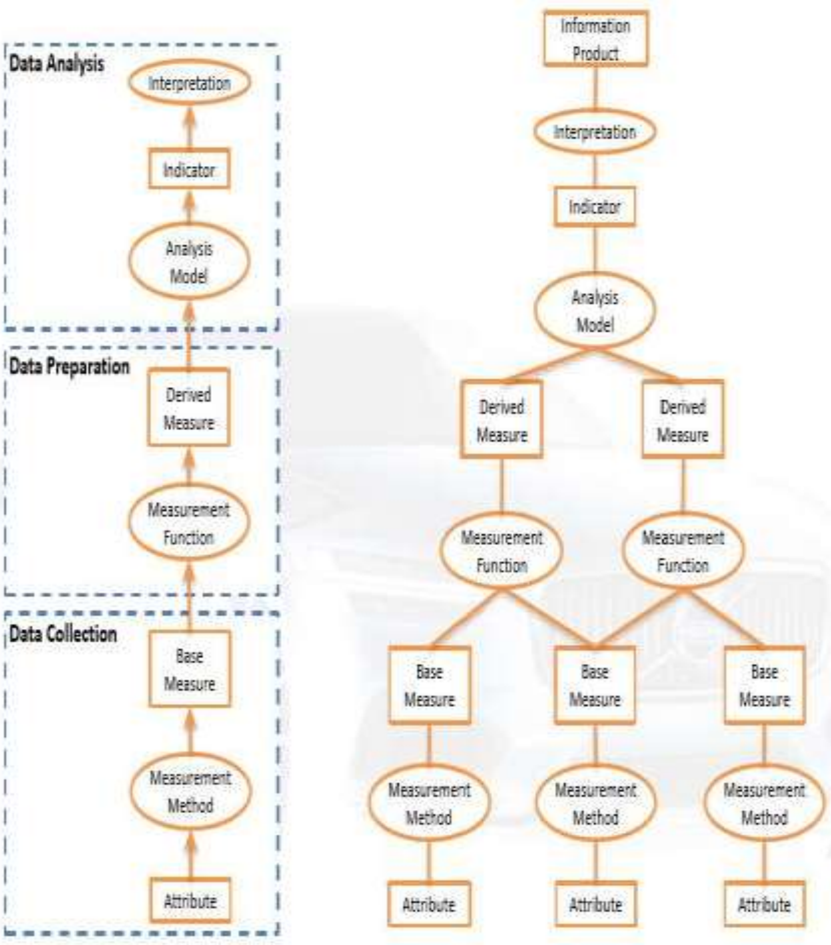
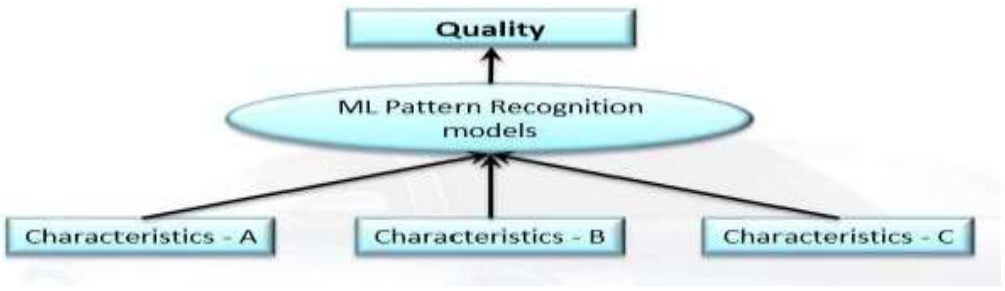
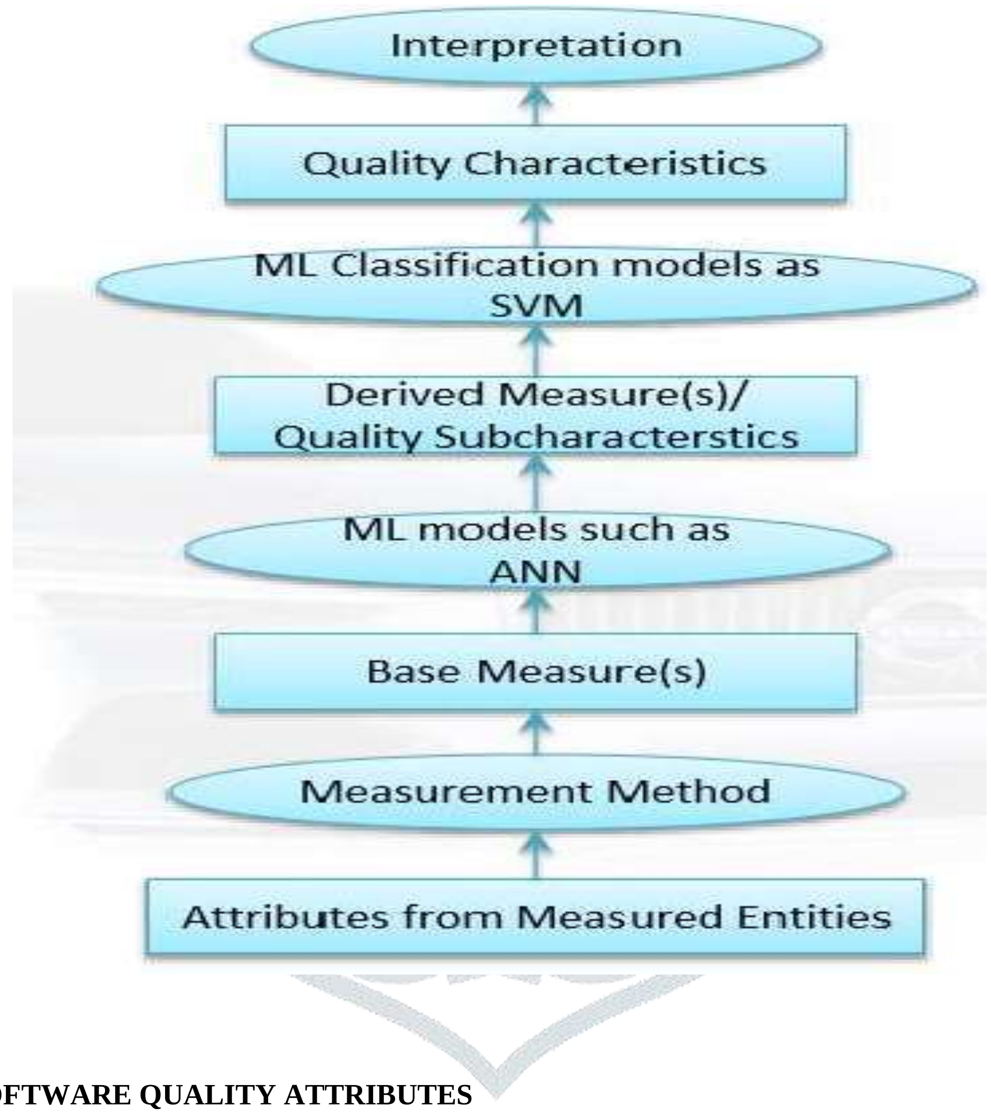


FIGURE 3- ISO 15939 Measurement Information

Framework for Quality Assessment using Ml



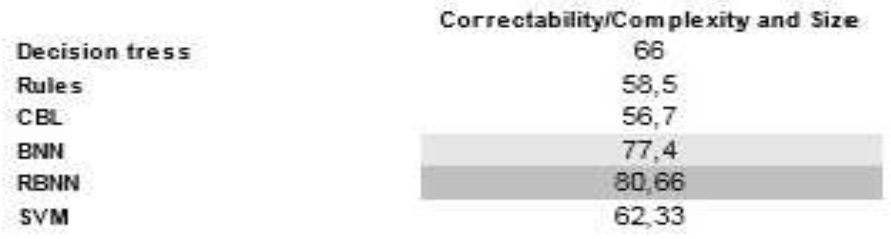

Interpretation
Quality Characteristics
ML Classification models as SVM
Derived Measure(s)/ Quality Subcharacterstics
ML models such as ANN
Base Measure(s)
Measurement Method
Attributes from Measured Entities

ASSESSING SOFTWARE QUALITY ATTRIBUTES

For building assessment models in the area of software product quality, we first have to connect internal measurable software artifacts, like, coupling, cohesion, inheritance, size, and complexity, to quality factors we want to assess. These internal artifacts are well documented in the literature and are presented for instance in [4] [5]

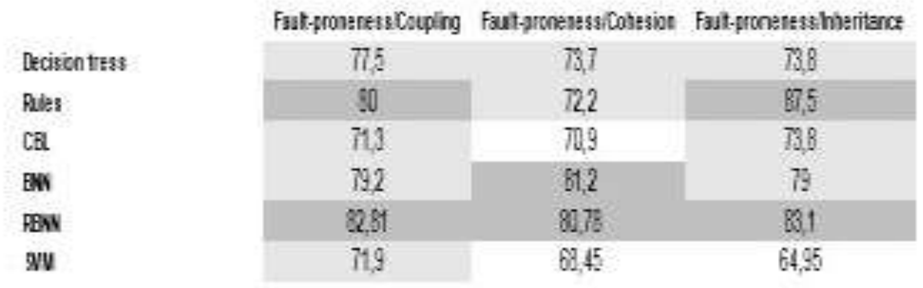
Maintainability is declined in 3 ways: (i) cost of corrective maintenance (correctability, the total effort spent to isolate and correct an ADA faulty component: low or high, see [4]), (ii) fault proneness at the class level (in a n component, here a C++ class, there was not any change of corrective type and, in a faulty one, there were one or more changes of corrective type during the development/maintenance phase, see [5]), and (iii) change impact, see [6]. On the other hand, reusability is defined in our context as the amount of work needed to reuse a component (a C++ class) from a system to another system of the same domain, i.e., the percentage of the code, which needs to be changed before reusing the component in a new system of the same domain. In order to generate assessment models, we have selected several algorithms belonging to the ML approaches presented in section 2. We have run them on software data collected from ADA, C++ and Java medium size applications. Some used ML algorithms are implemented in WEKA, an open source data-mining environment. We used also the BNJ tool (Bayesian Network tools in Java), which is written in Java and is available on the web1, and for the BP ANN, RBP ANN, and the SVM, we have used the MathLab tool. The computation of models accuracy is done thanks to a cross-validation procedure. It is helpful when the amount of data for training and testing is limited, which is our case. In terms of accuracy, and depending on which hypothesi algorithm we consider, we obtain very high results, especially for RBP ANN and BP ANN, and pretty high results for SVM and decision trees. Tables 1, 2, 3, and 4 illustrate these results.

TABLE I. ACCURACIES FOR CORRECTABILITY OF ADA COMPONENTS.



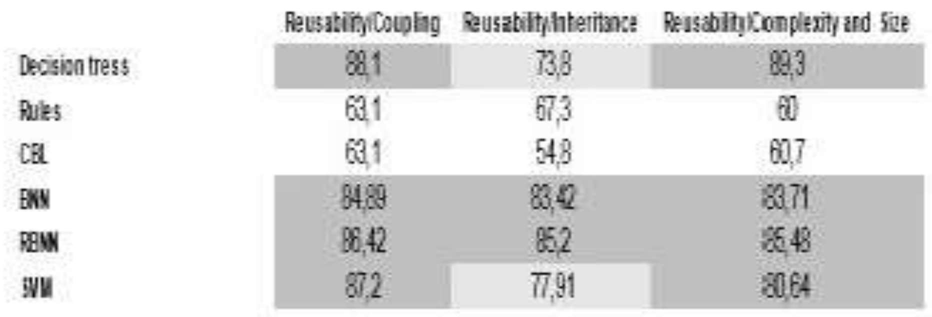
	Correctability/Complexity and Size
Decision tress	66
Rules	58,5
CBL	56,7
BNN	77,4
RBNN	80,66
SVM	62,33

TABLE II. ACCURACIES FOR FAULT-PRONENESS OF C++ CLASSES.



	Fault-proneness/Coupling	Fault-proneness/Cohesion	Fault-proneness/Inheritance
Decision tress	77,5	73,7	73,8
Rules	90	72,2	87,5
CBL	71,3	70,9	73,8
BNN	79,2	61,2	79
RBNN	82,81	80,78	83,1
SVM	71,9	68,45	64,95

TABLE III. ACCURACIES FOR REUSABILITY OF C++ CLASSES.



	Reusability/Coupling	Reusability/Inheritance	Reusability/Complexity and Size
Decision tress	88,1	73,8	89,3
Rules	63,1	67,3	60
CBL	63,1	54,8	60,7
BNN	84,89	83,42	83,71
RBNN	86,42	85,2	85,48
SVM	87,2	77,91	80,64

Firstly, in terms of selected internal metrics, we found certain uniformity between the different models. Decision trees and rules-based models; allow us to identify the internal software artifacts that are relevant for assessing a particular quality factor. For instance, The complexity/size metric NPM (number of parameters per method) is identified as a good indicator of reusability. In fact, combined with others like, NOP (number of polymorphic methods) and CSB (Class Size in Bytes), they allow us to determine if a component is reusable. It is also the case for design export coupling measured by OCAEC and OMMEC. On the other hand, the classic CBO metric (the number of other classes to which a class is coupled), and methods invocation, e.g., ACMIC (ancestor class method import coupling), RFC_α (the numbe methods that can potentially be executed in response to a message received by an object of that Firstly, in terms of selected internal metrics, we found a certain uniformity between the different based models, allow us to identify the internal software artifacts that are relevant for assessing a particular quality factor. For instance, the complexity/size metric (number of parameters per method) is identified as a good indicator of reusability. In fact, combined with others like, NOP (number of polymorphic methods) and CSB (Class Size in Bytes), they allow us to determine if a component case for design export coupling measured by OCAEC and OMMEC. On the other hand, the classic CBO metric (the number of other classes to which a class is coupled), and methods invocation, e.g., ACMIC (ancestor class method import coupling), RFC_α (the number of methods that can potentially be executed in response to a message received by an object of that (class), and IH-ICP (the number of ancestor methods invocation in a class, weighted by the number of parameters of the invoked methods) are identified as relevant coupling measures for fault proneness assessment. In the case of inheritance, NMI (the number of methods inherited), NMO (the number of overridden methods), and DIT (the maximum length from the class to a root) are the three that are stated relevant fault-proneness. Finally, LCOM2, LCOM5, and LCOM1 are the definitions of cohesion identified as relevant by the ML algorithms for assessing fault-proneness. Models induced for change impact analysis (table 4), especially trees and rul that among the selected metrics measuring coupling, five metrics are effectively relevant to change impact. Some of them are regarded as design metrics (AMMIC and OMMIC), others are considered as implementation metrics (MPC, CBOU, and CBONA). Rules produced by such glass-box models are exploitable to build a BN topology. After building the topology, we have to affect probabilities to the nodes. The entry nodes probabilities are directly deduced from measurements of these variables starting from data-set. Intermediate nodes are not directly measurable. They are defined or influenced by their parent nodes. Once the graph structure and all probabilities tables are defined, we can proceed with the Bayesian inference. Thus, BNs offer the possibility of processing scenarios of the form «What will occur if …?», allowing to identify potential problems and actions to be undertaken for improvement. For instance, by decreasing the metrics values CBONA and CBOU, change impact weakens more. Conversely, by CBONA and CBOU metrics values, the change impact becomes increasingly strong. These examples give an illustration of the interpretability and flexibility of the BNs models. They highlight the usefulness of such glass Moreover, and despite the fact that BP and RBP ANNs performances are close to 80% in table 1, we consider that these black allow us to conclude anything about the relevance of selected metrics for corrective maintenance costs assessment. By contrast, accuracy, rules-based models allow us to learn intelligible rules that identify relevant metrics. This is an example of an interpretation of such a rule: « one condition for an Ada faulty component to have a high corrective maintenance cost is when it is not well commented in comparison with the complexity revealed by the number of operators ». Of course, more experiments on more data extracted from various and representative systems are needed to confirm such conclusions.

7. MODEL SELECTION USING ML

POPULAR SOFTWARE DEVELOPMENT MODELS

A software cycle deals with various parts and phases from planning to testing and deploying software. All these activities are carried out in different ways, as per the needs. Each way is known as a Software Development Lifecycle Model (SDLC). [5] A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. The following are some basic popular models that are adopted by many software development firms .

3.1 The Waterfall Model When requirements are well defined and stable the waterfall model otherwise known as the classical life cycle, with its systematic and sequential approach can be utilized. The sequence begins with communication from the customer regarding specification and progresses through planning, modeling, construction and deployment. If the requirements are fixed and if work proceeds in a linear fashion to complete the project, then the waterfall model is appropriate.

3.2 Prototyping Model When detailed requirements for functions and features cannot be identified, and when the developer is not sure of the efficiency of an algorithm, the flexibility of an operating system, and the form of human-machine interaction, a prototype concept is utilized. “It is used as a technique that can be implemented within the context of any one of the process models.”[6] The prototype is made after fixing the overall objectives and requirements. The ensuing quick design climaxes in the construction of a prototype. The prototype is checked and refined with the feedback from the end users.

3.3 Rapid Application Development Model The Rapid Application Development Model (RAD) is an incremental software development process model that emphasizes a very short development cycle. The RAD model is a high speed adaptation of the waterfall model. The rapid development is achieved through component based construction. It results in a fully functional system within a very short period of time if the requirements are well understood and project scope is constrained. 3.4 Component Based Model The component based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature [7]. It makes intensive use of existing reusable components. The focus is on integrating the components rather than creating them from the beginning. The project cost and development cycle time can be reduced by incorporating component reuse as part of the organizational culture. The component based model has various steps ranging from requirements specification, component analysis, requirement modification, system design with reuse, development and integration and system validation.

SELECTING A SOFTWARE DEVELOPMENT LIFE CYCLE

Selecting a Software Development Life Cycle (SDLC) methodology is a challenging task for many organizations. Various software development life cycle models are suitable for specific project related conditions which include organization, requirements stability, risks, budget, duration of project etc. One life cycle model theoretical may suite particular conditions and at the same time other model may also looks fitting into the requirements but one should consider trade-off while deciding which model to choose. There are various methods employed in the industry to adopt a software development model that would be feasible to implement. Some of them maybe based on experience, expertise and even client demands. The crude numerical approach would help us adopt a software development model based on various characteristics of the project as given in [8] below mentioned points form the basis of adopting a software model

 Identify the characteristics of the project
 Score each available process model against the characteristics
 The method with the highest score wins
Here is a suitable checklist of characteristics:

- Are the requirements well-established, orill-defined? Interface?
- Are the requirements fixed, or likely to change as the project progresses?
- Is the project small to medium-sized (up to 4 people for 2 years) orlarge?

- Is the application similar to projects that the developers have experience in, or is it a new area? • Is the software likely to be is it straightforward or complex (e.g. does it use new hardware)?
- Does the software have a small easy user interface or a large complex user
- Must all the functionality be delivered at once or can it be delivered as partial products?
- Is the product safety critical or not?
- Are the developers largely inexperienced or mainly experienced?
- Does the organizational culture promote individual creativity and responsibility or does it rely on clear rules and procedures?

Identifying characteristics of project

Project Characteristic	**if true, score 1**	**if true, score 1**
requirements clarity	well-established	ill-defined
requirements change	fixed	changeable
project size	small to medium	large to huge
application	familiar	new
software	straightforward	complex
user interface	simple	complex
functionality	all at once	partial
safety critical	no	yes
developer expertise	largely inexperienced	largely experienced
culture	freedom	Order
User involvement	Minimal	Extensive
Project Characteristic	if true, score 1	if true, score 1
requirements clarity	well-established	ill-defined
requirements change	fixed	changeable
project size	small to medium	large to huge
application	familiar	new
software	straightforward	complex
user interface	simple	complex
functionality	all at once	partial
safety critical	no	yes
developer expertise	largely inexperienced	largely experienced
culture	freedom	Order
User involvement	Minimal	Extensive

Characteristics of Iterative model

Iterative model capabilities	**score**	**score**
requirements clarity	0	1
requirements change	0	1
project size	1	0
application	0	1
software	1	0
user interface	1	0
functionality	0	1
safety critical	0	1
developer expertise	0	1
User involvement	0	1
Total Score	**3**	**7**

Characteristics of waterfall model:

Waterfall model capabilities	**score**	**score**
requirements clarity	1	0
requirements change	1	0
project size	0	1
application	0	1
software	1	0
user interface	0	1
functionality	1	0
safety critical	0	1
developer expertise	0	1
User involvement	1	0
Total Score	**5**	**5**

8. CONCLUSION

The main objective of this work is to present assessment models based on ML techniques. The performances of these models vary, depending on the targeted quality factor. However, the first strong conclusion is that, with software projects data, black-box supervised approaches represented by SVM, and especially, BP and RBP ANNs, give constant and very high accuracies. The second conclusion concerns glass-box models and their ability to capture some explicit knowledge that will guide future software developments. The main strength of such ML models is that they are complementary. We can incorporate them in an automated decision-making process. However, such models need to be confirmed and generalized by more experiments on more software projects data, extracted from various and representative applications. This is the challenge to produce relevant and reusable models.

Software quality is the degree of conformance to explicit or implicit requirements and expectations. A software metric is a quantitative measure of a degree to which a software system or process possesses property with no defects. Hence, Software defect prediction model helps in early detection of defects using Classification Technique. In this paper we have discussed the various classification techniques such as Supervised, Un-supervised and Semi-supervised, which are applied on various datasets based on existing software metrics. In future we will be comparing the results of Supervised classification techniques on different datasets and open source projects to analyze the best classification technique to predict the defect in order to evolve a good software quality product.

• Quality in context of software is a common yet ambiguous term, • It varies based on the perspective and also on the environment in which a product is used and user expectations, • While software quality models and international standards have helped us understand better the factors that may affect quality, the relationship and effect size of individual factors/sub characteristics on overall quality is unknown, • Finding precise relationships is not only difficult but may well be impossible given that quality depends on number of characteristics internal, external and in use which in-turn can be affected by very large number of factors, • Large and mature software development organizations collects and monitors software metrics widely and is a part of their day today activities for all projects, this wealth of data can be effectively used internally to model software quality using traditional and ML based techniques.

We proposed a framework that uses machine learning techniques in conjunction with measurement information model of ISO/IEC 15939. • Using ML approaches means that we no need to know exact relationships between base and derived measures and build precise analysis model of how different quality sub characteristics affect higher order quality characteristics or overall quality. • Using the historical data, ML techniques can help assess overall quality and high order quality characteristics models based on measurable attributes. • Another very important benefit of using ML techniques is that they are self improving, thus as they are used in these large organizations and more data is collected over time, their accuracy and predictive power improves making them very attractive for such analysis.

Going through SDLC, popular software development models one can get awareness about the existing scenario. Some models based on experience, expertise and client needs vouch for their selection by developers. If one is sure of suitability of software development model characteristics to the project requirements, then it is easier to select an SDLC. The concept application illustrates and substantiates the validity of the chosen software development model. The questionnaire in the tabular format facilitates an easy framework and the interpretation of scores depending on software characteristics and project requirements enables a software developers.

REFERENCES

[1] Klopper, R., Gruner, S., & Kourie, D. (2007),0 ‖ Assessment of a framework to compare software development methodologies ‖ , Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, 56-65. doi:10.1145/1292491.1292498

[2] Software Methodologies Advantages & disadvantages of various SDLCmodels.mht

[3] Bennet, McRobb, and Farmer Object Oriented Systems Analysis and Design(Mc raw Hill 2002)p.46

[4] Bender RBT Inc., Systems Development Lifecycle: Objectives andRequirements

[5] Raymond Lewallen - CodeBetter.Com -Stuff you need to Code Better! Published08-01-2008

[6] Pressman Roger S., Software engineering pp- 33.

[7] Nierstraaz, O., s. Gibbs, and D. Tsichritziz, "Component – Oriented Software Development,"pp160-165

[8]
www.shu.ac.uk/.../project%20management%20-
%20new%20version.doc

9. Huda, Shamsul, et al. "A Framework forSoftware Defect Prediction and Metric Selection." IEEE Access(2017).

10. Ni, Chao, et al. "A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction." Journal of Computer Science and Technology 32.6 (2017):10901107.

11. Zimmermann, Thomas, et al. "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process." Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposiumon

The foundations of software engineering. ACM, 2009.

12. Laradji, Issam H., Mohammad Alshayeb, and Lahouari Ghouti. "Software defect prediction using ensemble learning on selected features." Information and Software Technology 58 (2015): 388-402.

13. Rajbahadur, Gopi Krishnan, et al. "The impact of using regression models to build defect classifiers." Proceedings of the 14th International Conference on Mining Software Repositories. IEEE Press,2017.

14. Yang, Xinli, et al. "TLEL: A two-layer ensemble learning approach for just-in-time defect prediction." Information and Software Technology 87 (2017):206-220.

15. Turhan, Burak, et al. "On the relative value of cross-company and within-company data for defect prediction." Empirical Software Engineering 14.5 (2009):540-578.

[16] L. Madeyski, M.Jureczko, "Which process metrics can significantly improve defect prediction models?", An empiricalstudy,(2014).

[17] D.Mehta, "A Comparative study of Techniques in Data Mining", by Manika Verma1 , International Journal of Emerging Technology and Advanced Engineering, vol. 4, no. 4,(2014).

[18] P. Reena, R. Binu, "Software Defect Prediction System –Decision Tree Algorithm With Two Level Data Pre-processing", International Journal of Engineering Research & Technology (IJERT), vol. 3, no. 3,(2014).

[19] G.Abaei, A.Selamat, "A survey on software fault detection based on different prediction approaches", Vietnam Journal of Computer Science, (2014), vol. 1, no. 2, pp.79-95.

[20] A.Okutan, O. T.Yildiz, "Software defect prediction using Bayesian networks", Empirical Software Engineering, (2014), vol. 19, no. 1, pp. 154-181. International Journal of Database Theory and Application Vol.8, No.3 (2015) 188Copyright

ⓒ 2015 SERSC [21] A.TosunMisirli, A. seBa¸

S.Bener,"A Mapping Study on Bayesian Networks for Software Quality Prediction", Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, (2014).

[22] R.Kalsoom, M. Qureshi, "Application andVerification of Algorithm Learning Based Neural Network",arXiv preprint arXiv:1406.2614, (2014), arxiv.org. [23] C. Catal, "A Comparison of Semi- Supervised Classification Approaches for Software Defect Prediction", Journal of Intelligent Systems, vol. 23, no. 1, pp. 75-82,(2013).

[24] R.Goyala, P.Chandraa, Y. Singha, "Suitability of KNN Regression in the Development of Interaction Based Software Fault Prediction Models", IERI Procedia, International Conference on Future Software Engineering and Multimedia Engineering, Elsiever, vol 6, pp. 15- 21,(2013),.

[25] G.Scanniello, C.Gravino, A.Marcus,T.Menzies,"Class level fault prediction using software clustering, Automated Software Engineering (ASE)", 2013 IEEE/ACM 28th International Conference,(2013).

[26] B. V. Balaji1, V.Venkateswara Rao2, "Improved Classification Based Association Rule Mining", International Journal of Advanced Research in Computer and communication Engineering, vol. 2, no. 5,(2013).

[27] R. M. Rahman, F. Afroz,"Comparison of Various Classification Techniques Using Different Data Mining Tools for Diabetes Diagnosis",Journal of Software Engineering and Applications, (2013), vol.6,pp.85-97

[28] T. Angel Thankachan1 , K. Raimond2 , "A Survey on Classification and Rule Extraction Techniques for Data mining",IOSR Journal of Computer Engineering ,vol. 8, no. 5,(2013), pp. 75-78.

[29] A. Chug1 and S. Dhall1 , "Software Defect Prediction Using Supervised Learning Algorithm and Unsupervised Learning Algorithm",The Next Generation Information Technology Summit (4th InternationalConference),(2013),pp.1-6.

[30] "Software defect prediction using supervised learning algorithm and unsupervised learning algorithm", Confluence 2013: The Next Generation Information Technology Summit (4th International Conference),(2013).

[31] M. Surendra Naidu, "Classification of Defects in Software Using Decision Tree Algorithm", International Journal of Engineering Science and Technology (IJEST),(2013).

[32] M. L., H. Zhang, R. Wu, Z.-H. Zhou, "Sample-based software defect prediction with active and semisupervised learning", Automated Software Engineering , (2012), vol. 19, no. 2, pp. 201-230

[33] H.Najadat and I.Alsmadi, "Enhance Rule Based Detection for Software Fault Prone Modules", International Journal of Software Engineering and Its Applications, vol. 6, no. 1, (2012).

[34] S. Kaur, and D. Kumar, "Software Fault Prediction in Object Oriented Software Systems Using Density Based Clustering Approach", International Journal of Research in Engineering and Technology (IJRET) vol. 1, no.2,(2012).

[35] K. Gao, T..M.Khoshgoftarr, "Software Defect Prediction for high- dimensional and class- imbalanced data", 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, (2011)Miami Beach, USA.

[36] B. Ma, D. Karel, V. Jan, B. Bart, "Software defect prediction based on association rule classification", Research Center for Management Informatics (LIRIS), Leuven,(2011).

[37] C. Catal, U.Sevim, B. Diri,"Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm", Elsevier,(2011).

[38] Y. Chen, P. Du,Xi , X.-H. Shen, "Research on Software Defect Prediction Based on Data Mining", Computer and Automation Engineering(ICCAE), 2nd International Conference,(2010),vol.1,pp.563-567.[39]T. [38] Y. Chen, P. Du,Xi , X.-H. Shen, "Research on Software Defect Prediction Based on Data Mining", Computer and Automation Engineering(ICCAE), 2nd International Conference,(2010),vol.1,pp.563-567.

[39]T.Nu Phyu, "Survey of Classification Techniques in DataMining", International MultiConference of Engineers and Computer Scientists, (2009); Hong Kong.

[40] C.-P.Chang a,*, C.-P.Chu a , Y.-F.Yehb , "Integrating in-process software defect prediction with association mining to discover defectpattern", Information and Software Technology ,vol. 51, no. 2, (2009), pp.375-384.

[41] D.Gray,D. Bowes, N. Davey, Y. Sun, "Bruce Christianson, Using the Support Vector Machineas a Classification Method for Software Defect Prediction with Static Code Metrics",11th International Conference, EANN 2009, (2009); London,UK.

[42] M. Jureczko, "Significance of Different Software Metrics in Defect Prediction", Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology,WybrzeżeWyspiańskiego vol. 27, pp.50-370.

[43] S. Lessmann,B.Baesens, C.Mues, and S. Pietsch,"Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings", IEEE Transactions on Software Engineering,(2008).

[44] S. Bibi, , G. Tsoumakas, I. Stamelos, I. Vlahavas, "Regression via Classification applied on software defect estimation",Elsiever,vol. 34, no. 3,(2008), pp. 2091-2101. [45] K. O. Elish, M. O. Elish, "Predicting defect-prone software modules using support vector machines" ,Elsevier, vol. 81,no. 5,(2008).