A Framework for Analyzing Software Quality using Hierarchical Clustering

Arashdeep Kaur
Deptt. Of CSE
Amity University
Noida, India

Sunil Gulati
Storage & Backup
HCL Comnet
Noida, India

Abstract— **Fault proneness data available in the early software life cycle from previous releases or similar kind of projects will aid in improving software quality estimations. Various techniques have been proposed in the literature which includes statistical method, machine learning methods, neural network techniques and clustering techniques for the prediction of faulty and non faulty modules in the project. In this study, Hierarchical clustering algorithm is being trained and tested with lifecycle data collected from NASA projects namely, CM1, PC1 and JM1 as predictive models. These predictive models contain requirement metrics and static code metrics. We have combined requirement metric model with static code metric model to get fusion metric model. Further we have investigated that which of the three prediction models is found to be the best prediction model on the basis of fault detection. The basic hypothesis of software quality estimation is that automatic quality prediction models enable verification experts to concentrate their attention and resources at problem areas of the system under development. The proposed approach has been implemented in MATLAB 7.4. The results show that when all the prediction techniques are evaluated, the best prediction model is found to be the fusion metric model. This proposed model is also compared with other quality models available in the literature and is found to be efficient for predicting faulty modules.**

Keywords- clustering; fault prediction; defect data; software quality; metrics.

I. INTRODUCTION

Software engineering discipline includes various prediction approaches such as effort prediction, cost prediction, fault prediction, reusability prediction, security prediction, maintainability prediction, and quality prediction etc. However, most of researchers are working on these prediction approaches but need for improvement is always there. Software fault prediction is the most popular research area in these prediction approaches and recently several research centers started new projects on this area. Software quality is one of the important facets to evaluate software. In literature various methods are available to predict software faults for evaluating software quality [2, 4, 9, and 13]. However, identifying and locating faults in software project is a difficult process. Today, more and more research is going on identifying faults in the early lifecycle of software development from previous releases or already developed similar kind of projects. Early software fault prediction can help the project managers to plan the desired resources, time and budget for better customer satisfaction. Several different techniques have been used to perform these tasks such as statistical methods, neural networks, machine learning etc. The increasing popularity and effectiveness of clustering algorithms have created possibilities for researchers to focus on building software fault prediction models. Fault prediction models operate on input datasets which contain metrics values extracted from various modules of the project such as Halstead's metrics, functional metrics, Mc. Cabe's Cyclomatic complexity etc. Fenton in his research show that using single features alone can be uninformative [3]. So, where individuals fail combinations can succeed. Hence, we have combined metrics available during requirement and code [2, 5]. The model is trained with available data from previous projects, the new data points are then inputted to classify the modules in to faulty and fault free.

The fault prediction model must be good enough to provide reliable product in available time frame and budget, meeting customer's requirements. However, various fault prediction model using clustering algorithms are already available in the literature but still there is a need to develop a robust model [1,14,15]. The goal of our research is to predict the fault prone areas in the software to be developed by using the data available with more accuracy.

Clustering is defined as the classification of data or object into different groups. It can also be referred to as partitioning of a data set into different subsets. Each data in the subset ideally shares some common traits. Data cluster are created to meet specific requirements that cannot created using any of the categorical levels. One can combine data objects as a temporary group to get a data cluster. Clustering can be agglomerative, fuzzy, hard, divisive, spectral, stochastic etc. But agglomerative is of our concern here.

In hierarchical clustering the data are not partitioned into a particular cluster in a single step. But a series of partitions takes place, which may vary from a single cluster containing all objects to n clusters each containing a single object. Hierarchical Clustering can be *agglomerative or divisive.* In this study, agglomerative clustering method is being used with Euclidean distance and complete linkage. Agglomerative clustering proceeds by series of fusions of the n objects into groups. Agglomerative techniques are used more commonly. An agglomerative hierarchical clustering procedure produces a series of partitions of the data, P_n, P_{n-1}, ..., P_1. The first P_n consists of n single object 'clusters', the last P_1, consists of single group containing all n cases. At each particular stage the method joins together the two clusters which are closest together (most similar). At the first stage, two objects are joined together that are closest, since at the initial stage each cluster has one object. In order to decide which clusters should be combined, a measure of dissimilarity between sets of observations is required. In most methods of hierarchical clustering, this is achieved by use of an appropriate metric (a measure of distance between pairs of observations), and a linkage criterion which specifies the dissimilarity of sets as a function of the pair wise distances of observations in the sets. Hierarchical clustering may be represented by a two dimensional diagram known as dendrogram which illustrates the fusions or divisions (in case of divisive) made at each successive stage of analysis.

The rest of the paper is organized as: Section II describes the methodology followed in detail. Section III throws light on the obtained results and Section IV will present the conclusions.

II. METHODOLOGY

Methods for identifying usability of software modules support helps to improve resource planning and scheduling as well as facilitating cost avoidance by effective verification. Such models can be used to predict the response variable which can either be the class of a module (e.g. faulty/fault free) or a quality factor (e.g. accuracy) for a module. However various software engineering practices limit the availability of fault data. With its effect supervised clustering is not possible to implement for analyzing the quality, usability, reusability, maintainability of software systems. This limited data can be clustered by using semi supervised clustering. It is a constraint based clustering scheme that uses software engineering expert's domain knowledge to iteratively create clusters as either usable or not. Software fault prediction models seek to predict two factors such as whether a component is fault free or not.

The methodology consists of the following steps:

A. Find the structural code and requirement attributes

The first step is to find the structural code and requirement attributes of software systems i.e. software metrics. The real-time defect data sets are taken from the NASA's MDP (Metric Data Program) data repository, available online at http://mdp.ivv.nasa.gov.in. In the MDP data, there are 13 projects, only 3 of them offer requirement metrics. The PC1 data is collected from a flight software system coded in C, containing 1107 modules and only 109 have their requirements specified. PC1 has 320 requirements available and all of them are associated with program modules. All these data sets varied in the percentage of defect modules, with the PC1 dataset containing the least number of defect modules.

B. Select the suitable metric values as representation of statement

The suitable metrics like product requirement metrics and product module metrics out of these data sets are considered.

The product requirement metrics are as follows:

- ***Module:*** This metric describes the unique numeric identifier of the product.
- ***Action****:* This metric represents the number of actions the requirement needs to be capable of performing.
- ***Conditional:*** This metric is used to represent whether the requirement will be addressing more than one condition.
- ***Continuance****:* This metric describe phrases such as the following: that follow an imperative and precede the definition of lower level requirement specification.

- ***Imperative****:* This metric describe those words and phrases that command that something must be provided.
- ***Option:*** This metric describe words that give the developer latitude in the implementation of the specification that contains them.
- ***Risk_Level:*** A calculated risk level metric based on weighted averages from metrics collected for each requirement.
- ***Source:*** This metric represent the number of sources the requirement will interface with or receive data from.
- ***Weak_Phrase****:* This metric describe clauses that are apt to cause uncertainty and leave room for multiple interpretations.

Table 1: Dataset Project Features

Project	Total Modules	Modules with requirement specified	Total Requirements	Requirements associated with program modules
CM1	505	266	160	114
PC1	1107	109	320	320
JM1	10878	97	74	17

The product module metrics are as follows:

- ***Module:*** This metric describes the unique numeric identifier of the product.
- ***Loc_Blank***: This metric describes the unique numeric identifier of the product.
- ***Branch_Count****:* This metric describes the branch count metrics i.e. the number of branches for each module.
- ***Call_Pairs****:* This metric describes the number of calls to their functions in a module.
- ***LOC_Code_and_Comment****:* This metric describes the number of lines which contain both code & comment in a module.
- ***LOC_Comments****:* This metric describes the number of lines of comments in a module.
- ***Condition_Count:*** This metric describes the number of conditionals in a given module.
- ***Cyclomatic_complexity****:* This metric describes the cyclomatic complexity of a module. It is the number of linearly independent paths.
- ***Cyclomatic_Density****:* This metric describes the ratio of the module's cyclomatic complexity to its length. The intent is to factor out the size component of complexity.
- ***Decision_Count****:* It describes the number of decision points in a given module. Decisions are caused by conditionals statements.
- ***Edge_Count****:* This metric describes the number of edges found in a given module. It represents the transfer of control from one module to another.
- ***Essential_Complexity:*** It describes the essential complexity of a module.
- ***Essential_Density****:* The Essential density is given by, (ev (G)-1)/ (v (G)-1) where ev (G) stands for essential complexity and v (G) stands for cyclomatic complexity.
- ***LOC_Executable****:* The number of lines of executable code for a module. This includes all lines of code that are not fully commented and blank.
- ***Parameter_Count****:* It describes the number of parameters to a given module.
- ***Global_Data_Complexity:*** Global Data Complexity quantifies the cyclomatic complexity of a module's structure as it relates to global/parameter data.
- ***Global_Data_Density****:* The Global Data density is calculated as: gdv (G) / v (G), i.e. dividing global data complexity by cyclomatic complexity.

- ***Halstead_Content****:* This metric describes the Halstead length content of a module.
- ***Halstead_Difficulty:*** The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program.
- ***Halstead_Effort:*** This metric describes the halstead effort metric of a module. Effort is the number of mental discriminations required to implement the program and also the effort required to read and understand the program.
- ***Halstead_Error_EST****:* This metric describes the halstead error estimate metric of a module. It is an estimate for the number of errors in the implementation.
- ***Halstead_Length:*** This metric describes the halstead level metric of a module i.e. level at which the program can be understood.
- ***Halstead_Prog_Time****:* This metric describes the halstead programming time metric of a module. It is estimated amount of time to implement the algorithm.
- ***Halstead_Volume:*** This metric describes the halstead volume metric of a module that contains the minimum number of bits required for coding the program..
- ***Normalized_Cyclomatic_Complexity:*** Normalized complexity is simply a module's cyclomatic complexity divided by the number of lines of code in the module. This division factors the size factor out of the cyclomatic measure and identifies modules with unusually dense decision logic. A module with dense decision logic will require more effort to maintain than the modules with less dense logic.
- ***Num_Operands:*** This metric describes the number of operands contained in a module.
- ***Num_Operators****:* This metric describes the number of operators contained in a module.
- ***Num_Unique_Operands:*** This metric describes the number of unique operands contained in a module. It is a count of unique variables and constants in a module.
- ***Num_Unique_Operators****:* This metric describes the number of unique operators contained in a module i.e. the number of distinct operators in a module.
- ***Number_Of_Lines:*** This metric describes the number of lines in a module. Pure, simple count from open bracket to close bracket and includes every line in between, regardless of character content. This metric describes the number of lines in a module. Pure, simple count from open bracket to close bracket and includes every line in between, regardless of character content.
- ***Pathological_Complexity:*** It describes the measure of the degree to which a module contains extremely unstructured constructs. Pathological complexity measures the degree to which a module contains extremely unstructured objects.
- ***Percent_Comments****:* This metric describes the percentage of the code that is comments.
- ***LOC_Total:*** This metric describes the total number of lines for a given module. This is the sum of the executable lines and the commented lines of code. Blank lines are not counted.

C. Analyze, refine metrics and normalize the metric values

In the next step the metrics are analyzed, refined and normalized and then used for modeling of fault prediction in software systems. PC1 static code based dataset contains 43 static code metrics of which various metrics that do not impact binary classification are removed like module identifier, call pairs, condition count, cyclomatic density, Decision count, Decision density, Edge count, Essential density, etc and the remaining 22 have been used for training and testing data. In the requirement based dataset the number of Input Metrics are eight.

D. Combine Requirement and code metrics

In Combining requirements and code metrics have done using inner join database operation. An inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. In Figure1 ER diagram relates the inner join between product_requirement_metric, product_requirement_relation and product_module_metric. The result of the join can be defined as the outcome of first taking the Cartesian product (or cross-join) of all records in the tables (combining every record in table A with every record in table B) - then return all records which satisfy the join predicate. Join operation is performed on product_module_metrics and product requirement relation using common attribute Module ID and result is

stored in temporary table. Then taking all records from temporary table and joining with product_requirement_metrics using Requirement ID.

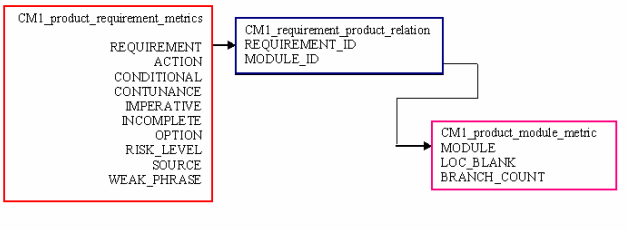

Figure 1. E-R Diagram relates project requirements to modules and modules to faults

E. Find the suitable algorithm for classification of the software components into faulty/fault free systems

In Clustering is the assignment of a set of observations into subsets (called *clusters*) so that observations in the same cluster are similar in some sense. Hierarchical clustering algorithms find successive clusters using previously established clusters. The algorithm is described as:

- Assign each item to a cluster so that if there are N items, we have N clusters.
- Find the closest pair of clusters and merge them in to a single cluster, so that now we have one cluster less.
- Compute distances between new cluster and the older clusters.
- Repeat step 2 and 3 until all the items are clustered in to a single cluster with N items.

In this paper the agglomerative hierarchical clustering method is being used also known as nearest neighbor technique.

F. Implementing the model and finding the result

The proposed approach has been implemented in MATLAB 7.4 environment. Clustering technique being used is a built-in function in its statistics toolbox. These functions are used with their default parameters.

To predict the results, we have used confusion matrix. The confusion matrix has four categories: True positives (TP) are the modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) are the fault-free modules correctly labeled as such. False negatives (FN) refer to faulty modules incorrectly classified as fault-free modules.

Table 2. A confusion matrix of prediction outcomes

Predicted	*Real data*		
		Fault	*No Fault*
	Fault	*TP*	*FP*
	No Fault	*FN*	*TN*

The following set of evaluation measures are being used to find the results:

- *Probability of Detection (PD),* also called recall or specificity, is defined as the probability of correct classification of a module that contains a fault.

$$PD = TP / (TP + FN) \quad (1)$$

- *Probability of False Alarms (PF)* is defined as the ratio of false positives to all non defect modules.

$$PF = FP / (FP + TN) \quad (2)$$

Basically, PD should be maximum and PF should be minimum [7]. ROC Curve has been plotted for result comparison after evaluating PD and PF values. A receiver operating characteristic (ROC) curve provides visual comparison of the classification performance. It can be represented equivalently by plotting the probability of

detection (PD) vs. probability of false alarms (PF). ROC analysis is related in a direct and natural way to cost/benefit analysis of software projects by comparing their detected defective and non-defective modules.ROC curves can be beneficial for finding accuracy of predictions. General ROC curve has a concave shape with (0, 0) as beginning and (1, 1) as end point.ROC curve is divided into different regions as shown in fig 3 below.

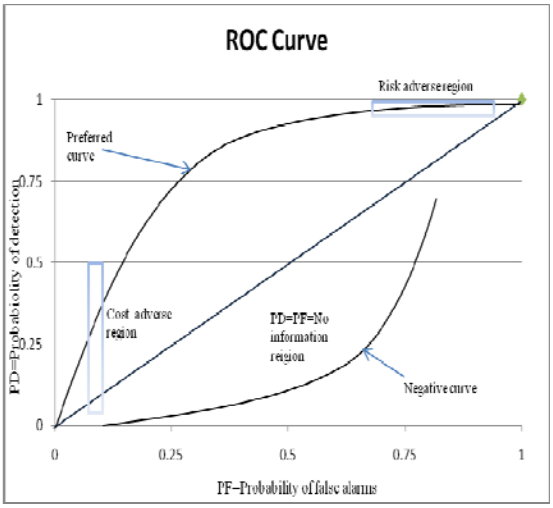

Fig.3. ROC curve

A straight line connecting the (0, 0) and (1, 1) implies that the performance of classifier is no better than random guessing thus this region is also known as no-information region. Risk adverse region indicates high PD and high PF, is beneficial for safety critical systems as identification of faults is more important than validating false alarms. Cost adverse region defines low PD and low PF, this region is beneficial for the organizations having limited Verification & Validation budgets. Negative curve with low PD and high PF can also be preferred for some of the software projects as it can be transposed into preferred curve when a classifier negates its tests. Preferred curve will include points that lie on or near to concave shape that starting from (0,0) an reached to (1,1) and cover both cost adverse region and risk adverse region .

III. RESULTS AND DISCUSSIONS

This study uses training and testing methodology, wherein a project is chosen for training the system, then a clustering algorithm is applied to the project and the final cluster centroids are then used to classify the modules of remaining projects as fault prone or fault free. The NASA MDP datasets are used in this approach to estimate the quality of a software product. This methodology is applied to all three models, code-based, requirement-based and fusion model with Hierarchical clustering algorithm. PD and PF values are obtained from the results which are then further used to plot ROC (Receiver Operator Characteristic) curves.

Table 3 Results of requirement metric model

PROJECT	CM1	PC1
TP	0	0
TN	21	213
FP	0	0
FN	68	107
PD	0	0
PF	0	0
Markers	C	D

Table 3 shows the results of using Hierarchical clustering on JM1 dataset as training data and using requirement metrics of CM1 and PC1 for testing to calculate their true positives, true negatives, false positives, false negatives, probability of detection and probability of false alarms. Here PD and PF as 0 and 0 respectively

indicate that performance is not much better. Table 4 gives the results of static code metrics models of CM1 and PC1. PD and PF values are 1 and 1 respectively for both CM1 code and PC1 code model. Here PD and PF as 1 and 1 indicates high PD and high PF, this such situation is preferred for safety critical systems because identification of faults is more important than cost to validate false alarms. Hence, these metrics can be effective if they can be combined.

Table 4 Results of code metrics model

PROJECT	CM1	PC1
TP	457	1031
TN	0	0
FP	48	76
FN	0	0
PD	1	1
PF	1	1
Markers	A	B

Table 5 shows the results of the fusion metric model extracted from requirement and static metrics. The results in terms of PD and PF for fusion metric model are more promising than the requirement metric model and the code metric model. As this model yield high PD and low PF.

Table 5 Results of combination metric model

PROJECT	CM1	PC1
TP	73	112
TN	15	3
FP	168	362
FN	10	0
PD	0.92803	1
PF	0.67952	0.96295
Markers	E	F

The information provided by the static and requirement metrics of all the three datasets is useless for project managers, as less number of the modules lie in the exact category either fault free/fault prone. The PD and PF of model derived from combination of requirement and code metrics is far accurate than other two models.

Fault-proneness models are models that are built from information about the code and its faults, and that relate code to faults. The existence of such classes of software would allow deriving fault-proneness models from historical data and then using such models for predicting fault-proneness of new software applications of the same class. Such models could be useful during both planning and executing testing activities. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and work. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities

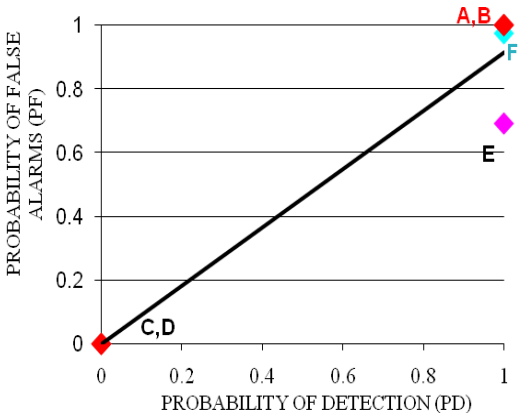

Fig.3. ROC curve for results

Fig. 3 is a graph which shows the comparison between different models on the basis of their PD and PF obtained and named as ROC. In figure, points A, B, C and D lie on the straight line, this means it lies on the no information region. Point E lies in high PD and low PF region i.e. Cost adverse region. Point F lies in High PD and high PF region i.e. risk adverse region, which is also preferable.

IV. CONCLUSIONS AND FUTURE WORK

We presented a fault prediction model using hierarchical clustering to estimate the software quality. In order to achieve a high quality development faults must be known prior to development so that more and smart emphasis can put in to that particular areas. Further we used the training and testing methodology. By analyzing the ROC curve performance of models can be drawn. Hence hierarchical clustering algorithm gives more accurate results than Fuzzy C Means algorithm [14]. This algorithm also produces good results in terms of fault prediction than K-means clustering algorithm [1]. To cope with a large number of tasks at hand, managers are always in search of a silver bullet that would give them a list of issues to focus their limited resources on. Hence this algorithm can be used to estimate software quality so that managers can plan the resources as and when required. In future we plan to investigate that which algorithm takes less number of iterations to provide more accuracy.

REFERENCES

[1] Kaur A., Sandhu P.S. and Brar A.S.(2009), "Early software fault prediction using real time defect data". In the proceedings of Second international conference on Machine Vision, Dubai, pp.242-245.

[2] Fenton N.E. and Pfleeger S.L. (1997), "Software Metrics: A Rigorous and Practical Approach". PWS publishing Company: ITP, Boston, MA, 2nd edition, pp.132-145.

[3] Jiang Y., Cukic B. and Menzies T. (2007), "Fault Prediction Using Early Lifecycle Data". ISSRE 2007, the 18th IEEE Symposium on Software Reliability Engineering, IEEE Computer Society, Sweden, pp. 237-246.

[4] Basu S., Banerjee A. and Moorey R.. (2002) "Semi-Supervised Clusering by Seeding". In Proceedings of the 19th International Conference on Machine Learning, Sydney, pp. 19-26

[5] Seliya N., Khoshgoftaar T.M. and Zhong S. (2005), "Analyzing software quality with limited fault-proneness defect data", in proceedings of the Ninth IEEE international Symposium on High Asssurance System Engineering, Germany, pp. 89-98.

[6] Bellini P. (2005), "Comparing Fault-Proneness Estimation Models", 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), China, pp. 205-214.

[7] NASA IV &V Facility. Metric Data Program. Available from http: //MDP.ivv.nasa.gov/.

[8] Lanubile F., Lonigro A., and Visaggio G. (1995) "Comparing Models for Identifying Fault-Prone Software Components", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, USA, pp. 12-19.

[9] Munson J.C. and Khoshgoftaar T.M. (1992), "The detection of fault-prone programs", IEEE Transactions on Software Engineering, vol. 18, issue: 5, pp. 423-433.

[10] Fenton N.E. and Neil M. (1999), "A Critique of Software Defect Prediction Models", IEEE Transactions on Software Engineering, vol. 25, issue: 5, pp. 675-689.

[11] M. Shepperd and D. Ince. A critique of three metrics. *The* Journal of Systems and Software, 26(3):197–210, September 1994

[12] Schneidewind, N. F. (2002). Body of knowledge for software quality measurement. IEEE Computer, 35(2), 77–83.

[13] Brodely C.E. and Friedl. M.A. (1999) "Identifying mislabeled training Data". Journal of Artificial Intelligence Research, vol. 11, pp.131-167.

[14] Kaur A, Sandhu P. and Brar A" An Empirical Approach for software fault prediction", Fifth International Conference on Industrial and Information Systems, pp.261-265, 2010..

[15] Puneet Jain, Arashdeep Kaur, "Software Fault Prediction with K-INF-means Clustering Using Normalized Early Lifecycle Data", proceedings of the International Conference on Advances and Emerging trends in Computing Technologies, pp. 131-135, 2010.