International Journal of
Science Engineering and Advance Technology

An Improved Cost Estimation in Software Project Development Using Neural Networks and COCOMOII model

Archana. R [1], Ravikumar.T [2],
[1]Student of M.Tech (CSE) and [2]Asst.Prof in Department of Computer Science Engineering,
AITAM, Tekkali, Srikakulam

Abstract:
An sympathetic of quality aspects is relevant for the software association to deliver high software dependability. An empirical consideration of metrics to prophesy the quality attributes is basic in order to acquire insight about the value of software in the primitive phases of software development and to certify corrective actions. Herein paper, we forecast a model to assess fault proneness via Object Oriented CK metrics and QMOOD metrics. We pertain one statistical method and six machine learning technique to predict the models. The proposed reproduction are validated using dataset unruffled from Open Source software. The consequences are analyzed using Area Under the Curve (AUC) achieve from Receiver Operating Characteristics (ROC) testing. The results show that the replica predicted using the random forest and bagging methods outperformed all the other mould. Hence, support on these results it is equitable to claim that quality models have a considerable relevance with Object Oriented metrics and that machine learning organizations have a equivalent performance with numerical methods. It is experimental that the CBR routine using the Mahalanobis detachment similarity occupation moreover the inverse distance weighted solution algorithm yielded the best fault prediction. In addition, the CBR models have superior performance than models basis on multiple linear regression.

Keywords: ***Software quality , Case-based reasoning , Software fault prediction.***

I. Introduction:

Experimental detection of fault-prone software mechanism enables verification professional to concentrate their time and resources on the obstruction areas of the system beneath development.

The facility of software quality forms to accurately identify essential components allows for the relevance of focused verification activities c from manual inspection to testing, dynamic analysis and static, automated formal analysis scheme. Software quality models, thus, help ensure the reliability of the delivered products. It has become imperative to develop and apply good software quality models early in the software development life cycle, especially in large-scale development efforts. Model performance comparison received attention in the software engineering literature (El-Emam et al. 2001). Nevertheless, empirical studies continue to apply different performance measures. Consequently, such studies do not encourage cross comparison with the results of work performed elsewhere. Many studies use inadequate performance metrics, those that do not reveal sufficient level of details for future comparison.

For these reasons, the objectives of this paper include:

1) A survey of frequently used model occurrence metrics and a discussion of their merits,

2) An foreword of cost curves, a new model estimate technique in software engineering, and 3) A comparison of model assessment techniques and a guide to their selection. We suppose that our findings and suggestion have a potential to enhance statistical legitimacy of future experiments and, ultimately, further the state of procedure in fault prediction modelling.

II.Related Work:

The target organization is a software purchaserside company that provides various types of telecommunication services using acquired software systems. In the software acquisition processes, the company is responsible for requirements analysis, architectural design, and acceptance testing, while developer-side companies are in charge of detailed design, programming unit/integration/ system testing, and debugging. As the services grow in the number of variations with shorter renewal cycles than ever before, the main motivation here is optimization of acceptance testing to provide high quality services to customers. From this perspective, the primary goal of this paper is reduction of acceptance test effort using techniques for predicting faultprone modules [7]. Our study includes metrics collection, building predictor models, and assessing the reduction of test effort.

C4.5 Algorithm:

The C4.5 algorithm is an inductive supervised learning system which employs decision trees to represent a quality model. C4.5 is a descendent of another induction program, ID3, and it consists of four principal programs: decision tree generator, production rule generator, decision tree interpreter, and production rule interpreter. The algorithm uses these four programs when constructing and evaluating classification tree models. Different tree models were built by varying parameters: minimum node size before splitting and pruning percentage.

The C4.5 algorithm commands certain preprocessing of data in order for it to build decision tree models. Some of these include attribute value description type, predefined discrete classes, and sufficient number of observations for supervised learning. The classification tree is initially empty and the algorithm begins adding decision and leaf nodes, starting with the root node.

Tree disc Algorithm:

The Tree disc algorithm is a SAS macro implementation of the modified CHi- square Automatic Interaction Detection algorithm. It constructs a regression tree from an input data set that predicts a specified categorical response variable based on one or more predictors. The predictor variable is selected to be the variable that is most significantly associated with the dependent variable according to a chisquared test of independence in the contingency table. Regression tree-based models are built by varying model parameters in order to achieve the preferred balance between the misclassification error rates, and to avoid over fitting of classification trees. A generalized classification rule is used to label each leaf node after the regression tree is built. This classification rule is very similar to the approach followed, when using S-PLUS regression trees as classification trees[8].

Sprint-Sliq algorithm:

Sprint-Sliq is an abbreviated version of Scalable parallelizable induction of decision Trees-Supervised Learning In Quest, the algorithm can be used to build classification tree models that can analyze both numeric and categorical attributes. It is a modified version of the classification tree algorithm of CART, and uses a different pruning technique based on the minimum description length principle. The algorithm has excellent scalability and analysis speed. Classification tree modeling using Sprint-Sliq is accomplished in two phases: a tree building phase and a tree pruning phase. The building phase recursively partitions the training data until each partition is either ‘‘pure’’ or meets the stop-splitting rules set by the user. The IBM Intelligent Data Miner tool, which implements the Sprint-Sliq algorithm, was used by our research group to build classification trees. Sprint-Sliq uses the Gini Index to evaluate the goodness of split of all the possible splits. A class assignment rule is needed to classify modules as fp and nfp.[9].

Logistic Regression:

Logistic regression is a statistical modeling technique that offers good model interpretation. Independent variables in logistic regression may be categorical, discrete or continuous. However, the categorical variables need to be encoded(e.g., 0, 1) to facilitate classification modeling. Our research group has used logistic regression to build software quality classification models. Let xj be the jth independent variable, and let xi be the vector of the ith module's independent variable values. A module being fp is designated as an ‘‘event’’. Let q be the probability of an event, and thus q=q/1-q is the odds of an event. The logistic regression model has the form [10], Log(q/1-q)=β0+β1x1+β1x2……….βjxj+βmxm Where, log means the natural logarithm, bj is the regression coefficient associated with independent variable xj, and m is the number of independent variables. Logistic regression suits software quality modeling

No. of faulty classes	281
% of faulty classes	63.57
No. of faults	500
Language used	Java

because most software engineering measures do have a monotonic relationship with faults that is inherent in the underlying processes. Given a list of candidate independent variables and a significance level, a, some of the estimated coefficients may not be significantly different from zero. Such variables should not be included in the final model.

III. Empirical Data Collection:

This study makes use of an Open Source dataset "Apache POI" [15]. Apache POI is a pure Java library for manipulating Microsoft documents. It is used to create and maintain Java API for manipulating file formats based upon the office open XML standards (OOXML) and Microsoft OLE2 compound document format (OLE2). In short, we can read and write MS Excel files using Java. In addition, we can also read and write MS word and MS PowerPoint files using Java. The important use of the Apache POI is for text extraction applications such as web spiders, index builders, and content management systems. This system consists of 422 classes. Out of 422 classes, there are 281 faulty classes containing 500 numbers of faults. It can be seen from Fig. 1 that 71.53% of classes contain 1 fault, 15.3 % of classes contain 2 faults and so on. As shown in the pie chart, the majority of classes consist of 1 fault. Table 2 summarizes the distribution of faults and faulty classes in the dataset.

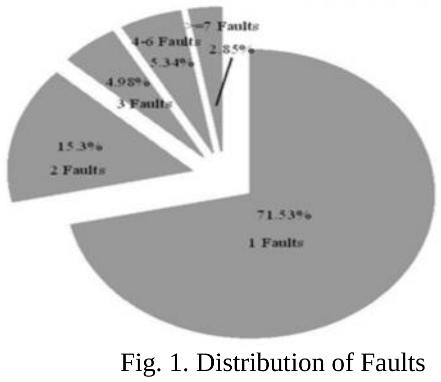

Fig. 1. Distribution of Faults

IV. Software Fault Proneness:

Software Fault Proneness is a key factor for monitoring and controlling the quality of software. The effectiveness of analysis and testing can be easily judged by comparing the predicted distribution of fault (Fault Proneness) and amount of fault found with testing (Software faultiness).

Fault Proneness of a class predicts the probability of the presence of faults in that class. Software analysis and testing are complex and expensive activities. Estimating and preventing the faults.

Early and accurately is better approach for reducing the testing efforts. If fault prone modules are known in advance, review, analysis and testing efforts can be concentrated on those modules.

V. Varying field defect occurrence patterns:

The field defect occurrence patterns vary greatly between different releases. Figures 4-7 provides a sample of four releases and their best post-facto fits.There are two implications. First, the diverse patterns mean that the Weibull model is best suited to model the field defect occurrence patterns since the Weibull is flexible enough to describe a wide range of patterns. This conjecture is supported by the fact that top two prediction methods use the Weibull model. However, the model parameters of the Weibull are harder to predict. The errors in predictions are exaggerated by the Weibull model form (in section 4.1 table 2); therefore, forecasts are not accurate. This is the same conclusion reached by Li et al. in [15]. Secondly, since the field defect patterns vary greatly between releases, more data are needed to distinguish between releases; however, we have limited data (at most 6 training data points) in our study.

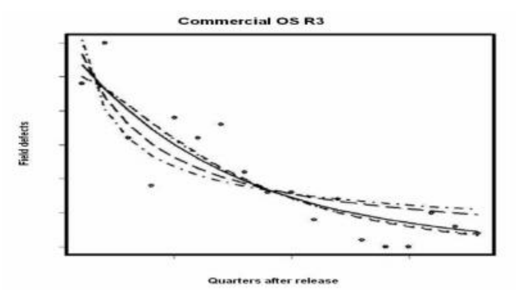

Fig. Actual field defects and fitted models for OS Release 3

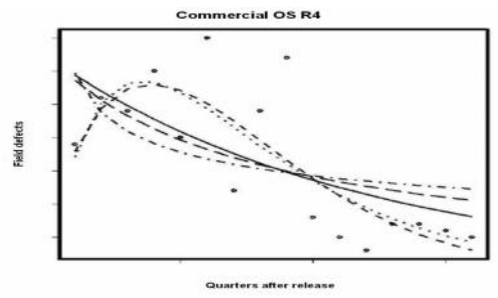

Fig. Actual field defects and fitted models for OS Release 4

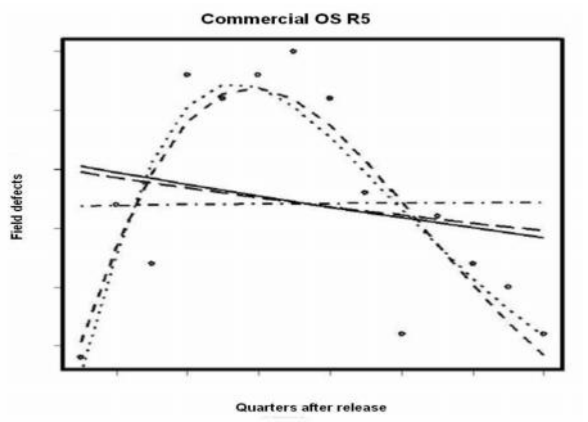

Fig. Actual field defects and fitted models for OS Release 5

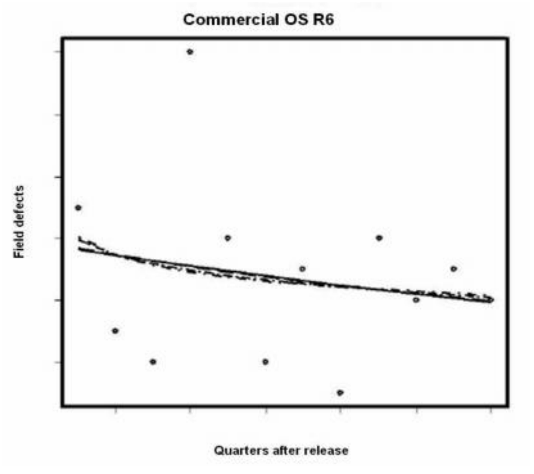

Fig. Actual field defects and fitted models for OS Release 6

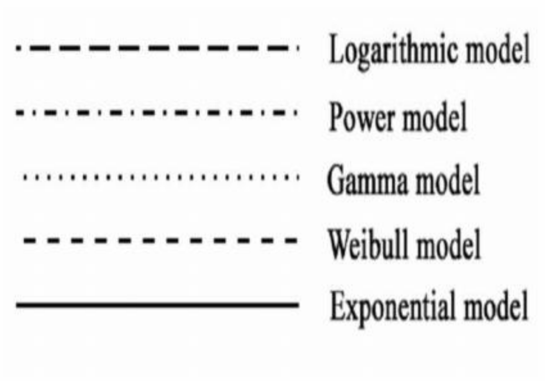

Fig. Legend for Figures 4-7

VI. Conclusion:

This paper reviewed software fault prediction papers published in conference proceedings and journals to evaluate the progress and direct future research on software fault prediction. We evaluated papers with a specific focus on types of metrics, methods, and datasets and did not describe all the prediction models in detail. The aim was to classify studies with respect to metrics, methods, and datasets that have been used in fault prediction papers. We evaluated papers published before and after 2005 with respect to metrics, methods, and datasets. We suggest the following changes in software fault prediction research:

*Conduct more studies on fault prediction models using class-level metrics. Even though object-oriented paradigm is widely used in industry, the usage percentage of class-level metrics are still beyond acceptable levels. We need prediction models using class-level metrics to predict faults during design phase and this type of prediction is called early prediction. In addition to classlevel metrics, the usage percentages of component-level and
process-level metrics are very low. It is an open research area to investigate component-level or process-level metrics for fault prediction problem.

*Increase the usage of public datasets for fault prediction problem. It is very significant to use public datasets for fault prediction because repeatable, refutable and verifiable models of software engineering can only be built with public datasets. However, he usage percentage of public datasets is 31% for this review and it is 52% for papers published after year 2005. Therefore, we need to increase the percentage of papers using public datasets and 80% can be an ideal level.

* Increase the models based on machine learning techniques. As specified in this review, machine learning models have better features than statistical methods or expert opinion based
approaches. Therefore, we should increase the percentage usage of the models based on machine learning techniques.

VII. REFERENCES:

[1] Assessing the Cost Effectiveness of FaultPrediction in Acceptance
Testing AkitoMonden, akuma Hayashi, Shoji Shinoda, KumikoShirai,
Junichi Yoshida, Mike Barker
[2] Predicting Defect Densities in Source Code Files with Decision Tree
Learners Patrick Knab, Martin Pinzger, Abraham Bernstein
[3] Comparative Assessment of Software Quality Classification
Techniques: An Empirical Case Study TAGHI M. KHOSHGOFTAAR,
NAEEM SELIYA
[4] A.S. Foulkes, Applied Statistical Genetics with R. Springer, 2009.
[5] A.L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate
Model for Software Reliability and Other Performance Measures," IEEE
Trans. Reliability, vol. 28, no. 3, pp. 206-211,Aug. 1979.
[6] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting Fault
Incidence Using Software Change History," IEEE Trans. Software Eng.,
vol. 26, no. 7, pp. 653-661, July 2000.
[7] "Information-Technology Promotion Agency, Japan (IPA) Software
Engineering Center (SEC) ed.," White Papers on Software Development
[8].CUCIS. Center for Ultra-scale Computing and Information Security, Northwestern University,
http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownloa
d.html
[9]. Yao, H., Hamilton, H.J., Geng, L.: A Unified Framework for Utility Based Measures for
Mining Itemsets. In: ACM SIGKDD 2nd Workshop on Utility-Based Data Mining (2006)
[10]. Pei, J.: Pattern Growth Methods for Frequent Pattern Mining. Simon Fraser University
(2002)
[11]. Sucahyo, Y.G., Gopalan, R.P.: CT-PRO: A Bottom-Up Non Recursive Frequent Itemset
Mining Algorithm Using Compressed FP-Tree Data Structure. In: IEEE ICDM Workshop
on Frequent Itemset Mining Implementation (FIMI). Brighton UK (2004)
[12]. FIMI, Frequent Itemset Mining Implementations Repository,
[13]. http://fimi.cs.helsinki.fi/
[14.] IBM Synthetic Data Generator, http://www.almaden.ibm.com/software/
quest/resources/index.shtml

Author:

Raghupatruni Archana is a student of Computer Science Engineering from Aditya Institute of Technology And Management, Tekkali, Presently pursuing M.Tech (CSE) from this college