International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 11
July 2016

A SYSTEMATIC APPROACH FOR REDUCTION OF TEST EFFORT USING FAULT PREDICTION MODELS

[1] M.SHAHNAZ PARVEEN, [2] U.DHANUNJAYA

[1] *M.Tech student, Department of CSE, Sri Krishnadevaraya University College Of Engineering & Technology, Ananthapuramu.*
[2] *Assistant Professor, Department of CSE, Sri Krishnadevaraya University College Of Engineering & Technology, Ananthapuramu.*

Abstract: Even though various approaches developed by the many researchers, they may not be optimal while predication of faults. In this approach we are introducing the fault prediction approach with OO metrics along with cyclomatic complexity and nested block depth, in acceptance testing, each function specified in the design document can be independently tested, that is, a set of test cases is developed for each function, not for each workflow module or other module/component. Our experimental results show the efficient fault prediction with our computation parameters.Our approach mainly concentrates on the count of faults prior to testing, expected number of faults, our classification which involves algorithmic and processing, control, logic and sequence, typographical Syntax errors i.e. incorrect spelling of a variable name, regular iteration of statements, incorrect initialization statements per module, this proposed classification approach shows optimal results while analyzing the metrics with training samples after measurement.

Index Terms: Metrics, Cyclomatic complexity, OO metrics, Fault prediction

I. INTRODUCTION

Software quality always depends on various factors like complexity, quality & size and it is not an end mark at customer satisfaction, even though customer satisfaction is the ultimate goal, quality of product is always basic feature of software development. Main developing software products, reusability is the basic important feature to use the existing code, it reduces the redundancy and complexity while execution of client request.

Traditional testing process is time consuming and expensive while handling of large projects . The traditional approaches are fault prediction works with the basic metrics like Lines of code(LOC),Number of errors found, Number of errors found with respect to the module, These parameters are not sufficient to measure the fault prediction and cost effectiveness. Software faults may be design faults which are deterministic in nature and are identified easily and other type of software faults is classified as being temporary internal faults that are transient and are difficult to be detected through testing.

It is difficult to analyze the fault prediction by simply measuring the software metrics of the project.we require a classification tool for the analysis of the predicted results.The use of software in high-assurance and mission-critical systems increases the need to develop and quantify measures of software quality. Subsequently, software metrics are useful in the timely prediction of high-risk components during the software development process such a prediction enables software managers to target quality improvement efforts to the needed areas. For example, prior to the system test, identifying the components that are likely to be faulty during operations can improve the effectiveness of testing efforts. Various software quality modeling techniques have been developed and used in real-life software quality predictions.

Classification-based modeling for software quality estimation is a proven technique in achieving better software quality. Some classification techniques used for software quality estimation include optimal set reduction, logistic regression, decision trees and neural networks, and case-based reasoning. Software metrics-based quality classification models classify software modules into groups.

A two-group classification model is the most commonly used, in which program modules are grouped into classes such as, fault-prone(f_p) and not fault-prone (nf_p).1 Software analysts interested in predicting whether a software component will be f_p or nf_p, can utilize classification models to obtain the needed prediction. When associating with such models, two types of misclassification errors are encountered: Type I error, that is, an f_p component is predicted as f_p, and Type II error, i.e., af_p component is misclassified as nf_p. Practically speaking, Type II errors are more severe in terms of software development costs and the organization's reputation. They may involve inspection and correction to components after they have been deployed for operations.

In our previous empirical studies related to software quality classification modeling, we have investigated several classification techniques, including classification and regression trees (CART), tree-based classification with S-PLUS, the Tree disc algorithm, the C4.5 algorithm , the Sprint-Sliq algorithm, logistic regression, and case-based reasoning.Classificationmodels

were calibrated using case studies of different large scale software systems, including the one presented in this paper. The models are calibrated to classify software modules as either f_p or nf_p, as defined by the system being modeled.

II.RELATED WORK

The target organization is a software purchaser-side 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 fault-prone 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 pre-processing 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 CH_i- 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 chi-squared 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 f_p and nf_p.[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 x_j be the j_{th} independent variable, and let xi be the vector of the i_{th} module's independent variable values. A module being f_p 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],

$$Log(q/1-q)=\beta_0+\beta_1x_1+\beta_1x_2\ldots\ldots\ldots\beta_jx_j+\beta_mx_m$$

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 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.

Architecture

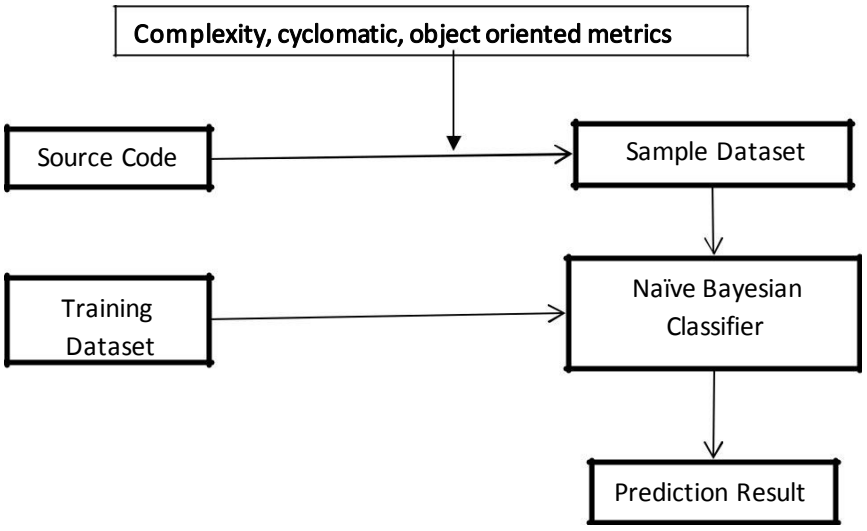

Fig 1: Proposed Architecture

III. PROPOSED WORK

We are proposing an efficient fault prediction with efficient parameters or metrics optimal results. The paper is concerned with faults due to Object oriented issues and measures with the integrated basic metrics of software engineering. Our proposed approach measures the metrics on the source code and analyzes the Testing samples which are measured Algorithmic and processing, Control, logic and sequence, Typographical Syntax error etc, then forwards these metric measures to the classification rule for analysis of testing sample, the below diagram shows proposed architecture, initially complexity, cyclomatic and object oriented metrics applied over source code blocks and measures respective parameters and treat them as testing samples and forward those testing samples to training samples to compute posterior probability of testing sample.

Basic Complexity metrics

Basic entity for measuring the software quality or cost are the metrics we initially computing the complexity metrics as Hallstead's metric, which includes program length, program vocabulary which leads to the computation of Estimated length and finally computes the purity ration, then computes the program effort in terms of volume and density.

n1 = the number of distinct operators

n2 = the number of distinct operands

N_1 = the total number of operators

N_2 = the total number of operands

From these numbers, several measures can be calculated:

Program vocabulary: $n=n_1+n_2$

Program length: $N=N_1+N_2$

Calculated program length:

$N' = {}_1 \log_2 {}_1 + {}_2 \log_2 {}_2$

Volume: $V=N+ \log_2$

It is a quantitative measure of logical strength of the program. It directly measures the number of linearly independent paths through a program's source code, these paths can be represented in various formats as sequence, while for iteration, if then else for decision making, until for iteration and then computes the complexity in terms of vertices or nodes and edges and in terms of number of

$$M = E - N + 2P,$$

where

program, with an edge between two basic blocks if control may pass from the first to the second. The complexity **M** is then defined as

E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

Object oriented metrics

Objected oriented metrics are also one type of metrics for measure the quality and cost of the project in terms of

Weighted methods per class
Depth of the inheritance tree
Coupling between classes
Response for a class
Class size and finally computes the polymorphism factor which includes the factors of number of overriding methods, number of methods and number of sub classes

Classification

After the computation of all the metrics, testing sample data can be forwarded to the training samples to measure and classify the metric results with existing results In terms of conditional probability by using Naive Bayesian classifier, which leads to the fault prediction analysis.

Naïve Bayesian Classification

Estimating probabilities

[1] Each data sample is of the type

$X=(x_i)$ i =1(1)n, where xi is the values of X for attribute A_i

2. Suppose there are m classes C_i, i=1(1)m.

$P(C_i|X) > P(C_j|X)$

i.e BC assigns X to class C_i having highest posterior probability conditioned on X

The class for which $P(C_i|X)$ is maximized is called the maximum posterior hypothesis.

From Bayes Theorem

3.P(X) is constant. Only $P(X|C_i)$, $P(C_i)$ need be maximized.

If class prior probabilities not known, then assume all classes to be equally likely
Otherwise maximize $P(C_i) = S_i/S$
Problem: computing $P(X|C_i)$ is unfeasible!

4.Naïve assumption: attribute independence

$P(X|C_i) = P(x_1,\ldots,x_n|C_i) = P(x_k|C_i)$

5.In order to classify an unknown sample X, evaluate for each class Ci. Sample X is assigned to the class C_i iff P

$P(X|C_i)P(C_i) > P(X|C_j)\ P(C_j)$.

Experimental Analysis

We experimentally analyzed the results by computing the all measures over code snippets individually and then computed parameters forwarded towards training dataset.

Let us consider an example for computing Halstead metrics for small code snippet as follows

```
main()
{
inta,b,c,avg;
scanf("%d %d%d", &a,&b,&c);
avg=(a+b+c)/3;
printf("avg=%d",avg);
}
```

The unique operators are main,(),(),int,scanf,&,=,*,/,printf
The unique operation are a,b,cavg,"%d %d %d",3,"avg=%d"

n_1=10, n_2=7, n =17
N_1=16, N_2=15, N=31
Calculated Program Length N'=10 × $\log_2$ 10 + 7 × $\log_2$ 7=52.9
Volume V=31 × $\log_2$17=126.7

Difficulty D=10/2 × 15/7=10.7
Effort E=10.7 × 126.7=1,355.7
Time required to program T=1,355.7/3000=0.004

For the measure of object oriented metrics, we considered weighted methods per class, Depth of the inheritance tree, Coupling between classes and overridden methods and for the measure of cyclomatic complexity we considering an example like

```
if A = 354
then if B >
C then A =
B else A=C
endif
endif
print A
```

Thus using the formal the cyclomatic complexity is 8-7 + 2 = 3. In this case there is no graph called or subroutine. Alternatively one may calculate the cyclomatic complexity using the decision points rule.

CONCLUSION:

We are concluding our research work with efficient metrics computation with Complexity, cyclomatic, object oriented metrics over blocks of source code followed by naïve Bayesian classification to compute whether the testing sample is cost effective or not by analyzing fault prediction.

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] T.M. Khoshgoftaar and E.B. Allen, "Modeling Software Quality with Classification Trees," Recent Advances in Reliability and Quality Engineering, pp. 247-270, World Scientific, 1999.

[5] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A.E. Hassan, "Revisiting Common Bug Prediction Findings Using Effort Aware Models," Proc. IEEE Int'l Conf. Software Maintenance, pp. 1-10, 2010.

[6] Y. Kamei, A. Monden, and K. Matsumoto, "Empirical Evaluationof SVM-Based Software Reliability Model," Proc. Fifth ACM/IEEEInt'lSymp. Empirical Software Eng., vol. 2, pp. 39-41, 2006.

[7] T.M. Khoshgoftaar and E.B. Allen, "Modeling Software Quality with Classification Trees," Recent Advances in Reliability and Quality Engineering, pp. 247-270, World Scientific, 1999.

[8] 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."Information-Technology Promotion Agency, Japan (IPA) Software Engineering Center (SEC) ed.," White Papers on Software Development Projects in Japan, 2010-2011 Ed., 2010.