*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. V (Nov – Dec. 2015), PP 31-35*
*www.iosrjournals.org*

# A Compound Metric for Identification of Fault Prone Modules

Ishleen Kaur
*Research Scholar, M.Tech (Computer Science Department, CDAC/GGSIPU, New Delhi, India)*

***Abstract:*** *Software Quality is a significant non-functional requirement that is not fulfilled by many software products. Faulty modules tend to degrade the software quality which may cause customer's dissatisfaction. Fault prone module prediction is an emerging area to improve the quality of the software and increase reliability of the system. Fault prone modules are predicted using various software metrics. In this paper, various metrics are investigated that have been used for fault prediction and the existing metrics are classified into three broad categories. Since no type of metric could accurately identify the fault prone modules, an integrated metric has been proposed to predict the fault prone modules.*

***Keywords:*** *Decision tree, fault prone modules, software maintenance, software metrics, software quality, software testing.*

## I. Introduction

Software maintenance in software engineering is the process of modification of the system after its delivery to correct the faults and improve performance of the system. It involves error correction, enhancements of capabilities, deletion of obsolete capabilities, and optimization. Thus, it may be corrective, adaptive, preventive or perfective. The key to maintenance is in development only. It can be encouraged by developing a high quality software and anticipating changes. But maintenance imposes some problems. It utilizes a lot of time and resources. For reducing the cost involved in software maintenance and improving the reliability of the system, it is essential to track the defects as early as possible in the software development life cycle. Due to the complexity and conditions under which a software is developed, it is becoming difficult to develop fault-free software. A faulty software module is the one containing faults that can cause software failure in an executable product. According to Pareto's principle, almost 20% of the software is responsible for 80% of the faults. The identification of the 20% of the software can help the developers and testers to focus on more critical modules. Thus, there is a need to detect the fault-prone areas of the software.

A fault prone module is a module which has the probability of fault being present. The accurate prediction of the fault prone modules can help direct cost, test effort and improve the software testing process, and thus quality of the system by focusing on fault-prone modules[1]. Thus, unnecessary testing of simple and non-faulty modules can be eliminated. It has been proved in the past studies that prediction of faulty modules provides a way to support software quality engineering through improved scheduling and project control[2].

The fault prone module prediction has been studied extensively in the past decades. Various techniques have been introduced to predict the fault prone modules[3-5]. Some of them use the code attributes like lines of code, Halstead software measures, cyclomatic complexity for the development of mathematical models[5-6]. There are various software metrics that have been used for the defect prediction. Software metric is a quantitative measure of degree to which a system possesses a particular attribute. Metrics from various phases of the life cycle have been used in previous studies for prediction of fault prone modules. There are studies which use the requirement metrics in combination with the code metrics to predict the fault prone modules[7-8]. Though requirement metrics alone has no significance in predicting fault prone modules, the combination of requirement metrics and code metrics achieved high prediction results. The UML diagrams are created at the design phase of the software development life cycle and the metrics extracted from these UML diagrams are used for the prediction of fault prone modules[9-11]. Metrics from source code have proved to be a good source for prediction of fault prone modules. Various studies involve the use of CK metrics for prediction of fault prone modules[12-13]. CK metrics are the object oriented metrics proposed by Chidamber and Kemerer[14][15]. These metrics were designed to measure unique aspects of the OO approach and measure the complexity of the design. But no metric can predict the fault prone modules accurately. Thus, there is a need to introduce a new metric for the fault prone module prediction.

The aim of this study is to study different metrics and techniques that have been used for identification of fault prone modules. All the metrics for prediction of fault prone modules have been classified into three broad categories. A new compound metric is introduced to improve the fault-prone modules identification process. The proposed metric uses the metric from various phases of the software development life cycle.

The rest of the paper is organized as follows: Chapter II describes the types of metrics for fault prone module identification. In chapter III, related work is presented. Chapter IV presents the proposed approach of this study, while chapter V describes the conclusion of the study.

## II. Types Of Metrics For Fault Prone Module Identification

Software metrics are used as the characteristics of a module for the fault prediction. There are broadly three types of metrics to identify the fault prone modules which are shown in Table I.

**Table I.** Types of Metrics

| Type | SDLC Phase |
|---|---|
| Requirement Metrics | Requirement |
| Design Metrics | Design |
| Code Metrics | Implementation |

### 1.1 Requirement Metrics

The requirement metrics are those metrics that have been extracted from the textual characteristic requirements specification of the system. Since majority of the defects are introduced in the system in the requirement phase, due to misinterpretation of requirements or missing requirements etc, requirement metrics are an important source for the prediction of fault prone modules.
The requirement metrics used in the past studies[7-8] are shown in Table II.

**Table II.** Requirement Metrics

| | |
|---|---|
| ACTION | The total number of actions that the requirement will address. |
| CONDITIONAL | It determines whether the requirement will address more than one condition. |
| CONTINUANCE | It determines the connections between statements. |
| IMPERATIVE | It determines the important requirements that must be addressed. |
| INCOMPLETE | It represents the requirements that are yet to be determined. |
| OPTION | It represents the requirements which give the developers latitude in implementing them. |
| RISK_LEVEL | It represents the risk level of requirements. |
| WEAK_PHRASE | It represents the requirements that leave multiple interpretations. |
| SOURCE | It represents the number of sources that the requirement will interface with. |

The requirement metric set alone is not sufficient for the identification of fault prone modules as it is too early to predict the fault prone modules. The faults can be introduced in any phase of the life cycle of the system, so the faults introduced after the requirement phase will not be taken into account in the prediction model.

### 1.2 Design Metrics

The design metrics are extracted from the various UML diagrams of the system that have been prepared in the design phase of the life cycle. These metrics can be extracted from the software design document of the system. The design document depicts the architectural design and modeling artifacts of the system. Various UML diagrams like statechart diagram, component diagram, sequence diagram etc are used to identify the fault prone modules by analyzing the architecture of the system.
Some of the design metrics used in the past studies[9-11] are shown in Table III.

**Table III.** Design Metrics

| | |
|---|---|
| Decision count | The number of decision points in a module. |
| FanIn | The number of receive signal objects. |
| FanOut | The number of send signal objects. |
| Signal | The number of signals used to communicate with other modules. |
| Arcs | The number of arcs used to connect to other nodes. |
| Paths | The number of paths in a module. |
| Branches | The number of branch count in a module. |
| Node Count | The number of nodes in a module. |

But the architectural analysis of the system is not enough to identify the fault prone modules. The architecture of an application can be simple but can have a very complicated internal logic of procedures. This complicated internal logic will be ignored while identifying fault prone modules.

### 1.3 Code Metrics

The code metrics are extracted from the source code of various modules of the system. These metrics process the source code of a component and estimate the metrics based on code specifications such as number of methods in a class, branch statements etc.
Some of the code metrics used for the identification of fault prone modules are shown in Table IV:

**Table IV.** Code Metrics

| | |
|---|---|
| LINES OF CODE(LOC) | The total number of source lines of code |
| NUMBER OF CHILDREN(NOC) | The number of immediate subclasses of a class. |
| WEIGHTED METHODS PER CLASS(WMC) | The summation of the complexities of methods defined in each class. |
| DEPTH OF INHERITANCE(DIT) | The maximum depth of class in the inheritance tree. |
| COUPLING BETWEEN OBJECTS(CBO) | The number of classes to which a particular class is coupled. |
| RESPONSE FOR A CLASS(RFC) | The number of methods which can be invoked in response to a message to the object. |
| LACK OF COHESION OF METHODS(LCOM) | The dissimilarity of methods in a class. |
| CYCLOMATIC COMPLEXITY | The independent paths in a module. |
| NUMBER OF COMMENTS | The total number of lines of comment in the code. |
| NUMBER OF OVERRIDDEN METHODS | The total number of methods a module overriddes. |
| NUMBER OF OPERANDS | The total number of operands used in a module. |
| NUMBER OF OPERATORS | The total number of operators used in a module. |
| LINES OF COMMENTS | The number of lines of comments written in a module. |

Code metrics are essential for identifying the fault prone modules as the source code of the system includes almost all of the faults introduced in the system. But just using code-based metrics will not generate a high prediction model as the relation between the components of the system can be complicated which cannot be taken into account using the code metrics.

## III. Related Work

There have been a lot of studies to predict the fault prone modules. The design metrics have been used for the fault prediction using various UML diagrams. Ohlsson et al[11] used the design metrics for the prediction of fault prone modules. It presents a case study of Ericsson Telecom AB to predict fault prone modules at the design phase. In this paper, a method for predicting the most fault prone modules is proposed which is done by ranking using Spearman's test to determine the correlation between various design metrics and observed costs of handling trouble reports. The results concluded that it followed the Pareto's principle identifying 20% of the source code responsible for 50-60 percent of the faults.

Singh and Verma[16] used the metrics computed at design phase to predict the fault prone modules. The design metrics are used for predicting modules of target project using other projects as training projects. The machine learning technique used for evaluation is Naïve Bayes. In total, 7 projects from NASA MDP repository have been used for the fault prediction. Almost all the cross projects achieved AUC results of more than 0.7.

Sandhu et al[8] used the combination of requirement metrics and code metrics for the prediction of fault prone modules. The reason for the integrated approach is that though requirement metrics alone are not good predictors for predicting the fault prone modules, they can be combined with the code metrics to improve the prediction results. The data set of CM1 project has been used for the experiment, collected from NASA Metrics Data Program (MDP) data repository. In total, 31 metrics have been used in this approach. For classification of modules, decision tree based model is combined with k-means clustering. After clustering, decision Tree C4.5 algorithm along with 10-fold cross validation method is used to measure the performance of the model. The experiments predicted with 100% precision and 100% recall. But the modules used for the experiment are very less due to the lack of availability of data.

Shanthini and Chandrasekaran[17] used the code metrics for the prediction of fault prone modules. These code metrics have been collected from the NASA MDP datasets. For the classification, support vector machine has been used in combination with an ensemble method, bagging. Support Vector Machine(SVM) constructs an N-dimensional hyper plane which optimally separates data into fault prone and non-fault prone classes. Bagging method has been used to improve the prediction results. It was concluded that bagged SVM classifier model performs better than the baseline classifier model across different level of software metrics.

Gaur and Kaur[18] analysed the relationship between faulty classes and the code metrics using six machine classifiers. Machine learning algorithms are the classifiers that predict the fault prone classes using training data. The classifiers used in this study include logistic regression, naïve bayes, instance based, bagging, decision tree and random forest. The results concluded that random forests provided best ROC for most of the projects.

Bishnu and Bhattacherjee[19] applied a quad tree based k-means algorithm for predicting fault prone modules. In this paper, quad trees are applied for finding initial cluster centers for K-Means algorithm. Then the quad tree based algorithm is applied for predicting fault prone modules in the software. A Quad Tree in two dimensional space is a 4-way branching tree which represents recursive decomposition of space. At each level, each square is divided into four equal size squares. It solved the disadvantages of k-means clustering algorithm which are selection of the initial cluster centers and sensitivity to noise. The technique is implemented on six projects. The results concluded that the QDK algorithm works as an effective initialization algorithm. The number of iterations of k-Means algorithm is less in the case of QDK algorithm in most of the cases.

Aman[20] used the lines of comments to predict the fault proneness of small size modules. Comment lines include description of code, usage of functions or modules. Code lines are written in a specific programming language, but comment lines are written in a free form. Comments have been considered a size independent metric in the prediction. The idea behind using comments is that comments written in a small size module is a sign of complex module. Two types of comments are used in the paper: block comments and end of line comments. The approach is implemented on a Java project and the result shows that a module in which some comments are written is more likely to be faulty than non-commented ones.

## IV. Proposed Approach

It has been observed from the study that the requirement metrics, design metrics or the code metrics cannot be alone used for the fault prediction. Thus in the proposed approach, a new compound metric is introduced which can be used for the fault prediction process including the metrics from all the three phases discussed to cover the drawbacks found in each type of metric. We will build models to characterize the documents obtained from each development phase and use them in fault prediction process.
The methodology followed in the proposed approach is given in Fig.1.

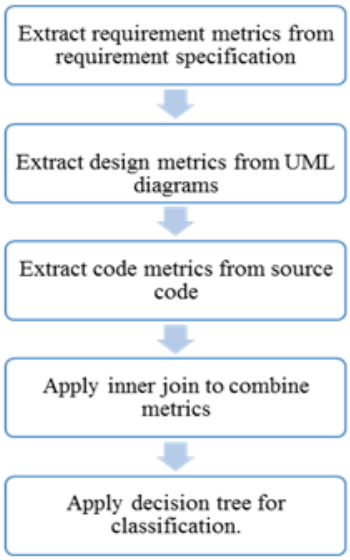


**Figure 1.** Methodology of proposed approach

Extract the requirement metrics: The requirement specification document is used to extract the various requirement metrics. Many of the severe faults are introduced into the system at the requirement phase. So, requirement metrics can be used to improve the fault prediction process. The requirement metrics used in the proposed approach are: Action, Conditional, Continuance, Imperative, Incomplete, Option, Risk Level, Weak phrase and Source.

Extract the design metrics: UML diagrams depicts the architecture of the system. The metrics computed from the design document helps capture the faults introduced at the architecture level of the system. The design metrics used in the proposed approach are: Decision count, Branch count, arcs, node count, fan-in, fan-out and cyclomatic complexity.

Extract the code metrics: The metrics from source code captures the internal complexity of each module. The metrics from source code used in the proposed approach are: lines of code, weighted methods per class, response for a class, number of children, coupling between objects, depth of inheritance and lack of cohesion of methods.
So, in total 23 metrics from various phases are used in the proposed approach.

Combine metrics using inner join: After extracting the metrics from the requirement specification, design document and source code of the system, inner join method is applied. This step is necessary to link the metrics from all phases for each module. Also, each module does not have the metrics from all the phases, the modules having metrics from all the three phases are used in this study.

Decision tree algorithm for classification: The machine learning algorithm is applied for learning from the training data sets. The training data is required from the past history, having details about the fault data along with the document from requirement phase, design phase and source code of the system. Decision tree algorithm is used for the classification. Decision tree represents rules. By following the tree, the rules can be used to

understand why a record is classified in a certain way. The question at the root must be the one that best differentiates among the target classes. The leaf node determines the classification of the record.

The proposed approach is a supervised learning technique. Thus it requires training data including the project and the fault data indicating faulty and non-faulty classes. Using the data from training project, decision tree can be optimally used to predict the fault prone classes of the target project.

## V. Conclusion

The fault prone module prediction is an emerging area of research. The correctly classified fault prone software modules can be used to optimize available resources and time. In this paper, we have studied various metrics and techniques for the identification of fault prone modules. The metrics are classified into three categories: requirement metrics, design metrics and code metrics. Since only one type of metric cannot provide high accuracy in prediction, a compound metric involving metrics from all the phases of life cycle is introduced in this paper. The classification is achieved by a machine learning approach, decision tree algorithm. The proposed metric aims to provide high prediction results than existing techniques. In future, the applicability and reliability of the approach will be evaluated with the help of open source projects.

## References


[1]. C. Catal, "Software fault prediction: A literature review and current trends" *Expert Systems with Applications*: *An International Journal, 38(4),* 2011, 4626-4636.

[2]. P.S. Sandhu, S.K. Dhiman and A. Goyal, "A Genetic Algorithm Based Approach for Classification of Fault Prone Modules", *in International Journal of Computer, Electrical, Automation, Control and Information Engineering*, World Academy of Science, Engineering and Technology, *3(12),* 2009.

[3]. K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, *Journal of Systems and Software, 81(5),* 2008, 649– 660.

[4]. D. Kaur et al, "A Clustering Algorithm for Software Fault Prediction", *16th IEEE International Conference on Computer and Communication Technology(ICCCT),* 603-607, 2010.

[5]. C. Jin et al, "Artificial neural network-based metric selection for software fault-prone prediction model", *The Institution of Engineering and Technology*, 6(6), 2012, 479-487.

[6]. A.K. Pandey, N.K. Goyal, "Test Effort Optimization by Prediction and Ranking of Fault-prone Software Modules", in *International Conference on Reliability, Safety & Hazard*, 2010, 136-142.

[7]. Y. Jiang, B. Cukic and T. Menzies, "Fault Prediction Using Early Lifecycle Data" *ISSRE, 18th IEEE Symposium on Software Reliability Engineering*, 2007, 237-246.

[8]. P. S. Sandhu, A.S. Brar, R. Goyal and J. Kaur, "A Model for Early Prediction of Faults in Software Systems", *2nd IEEE International Conference on Computer and Automation Engineering*, 4, 2011, 281-285.

[9]. S. Wagner and J. Jurjens, "Model Based Identification of Fault-Prone Components", *5th European Dependable Computing Conference*, Springer, Budapest, Hungary, 2005, 435-452.

[10]. Marshima M. Rosli, Noor Hasimah Ibrahim Teo, Nor Shahida M. Yusop and N. Shahriman Mohamad, "Fault Prediction Model for Web Application using Genetic Algorithm" *in International Conference on Computer and Software Modeling(IPCSIT),* vol.14, 2011.

[11]. N. Ohlsson, M. Helander and C. Wohlin, "Quality Improvement by Identification of Fault-Prone Modules using Software Design Metrics", *6th International Conference on Software Quality*, Ottawa, 1996.

[12]. Arvinder Kaur and Ruchika Malhotra, "Application of Random Forest in Predicting Fault-Prone Classes", in *International Conference on Advanced Computer Theory and Engineering(ICACTE),* pp. 37-43 Pukhet, 2008.

[13]. H. Kundra, A. Sharma and R.P.S. Bedi, "Evaluation of Fault-Proneness of Modules in Open Source Software Systems Using k-NN Clustering", in *International Journal of Engineering Sciences, vol.4*, 2011

[14]. Shyam R. Chidamber and Chris F. Kemerer. "Towards a metrics suite for object oriented design", *Proc. 6th ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA),* 1991, 197–211.

[15]. Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Trans. on Software Engineering, 20(6),* 1994, 476-493.

[16]. P. Singh and S. Verma, "Cross Project Software Fault Prediction at Design Phase", *in International Journal of Computer and Information Engineering*, World Academy of Science, Engineering and Technology, vol.2, no.3, 2015.

[17]. A. Shanthini and RM. Chandrasekaran, "Analyzing the Effect of Bagged Ensemble Approach for Software Fault Prediction in Class Level and Package Level Metrics, in *IEEE International Conference on Information Communication and Embedded Systems(ICICES)*, 2014, 1-5.

[18]. S. Gaur and S. Kaur, "Empirical Study on Software Quality Models: NASA MDP", *in Journal of Computer and Communications, 1(5),* 2013.

[19]. P.S. Bishnu and V. Bhattacharjee, "Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm, *in IEEE Trans. on Knowledge and Data Engineering, 24(6),* 2012.

[20]. H. Aman, "An Empirical Analysis of the Impact of Comment Statements on Fault-Proneness of Small-Size Module", *19th IEEE Asia Pacific Software Engineering Conference(APSEC),* 1, 2012, 362-367.