A Survey on Software Fault Detection based on Different Approaches

Sri Gorantla
University of Akron
sg303@uakron.edu

Abstract

One of the software engineering interests is quality assurance activities such as testing, verification and validation, fault tolerance and fault prediction. When any company does not have sufficient budget and time for testing the entire application, a project manager can use some fault prediction algorithms to identify the parts of the system that are more defect prone. There are so many prediction approaches in the field of software engineering such as test effort, security and cost prediction. Since most of them do not have a stable model, software fault prediction has been studied in this paper based on different machine learning techniques such as support vector machines, Genetic Algorithms , OneR ,Naïve Bayes and another method called Fuzzy model which uses Software metrics and Process Maturity. This paper is a survey on four different papers that explains different approaches for Software Fault Detection.

I. INTRODUCTION

As today's software grows rapidly in size and complexity, the prediction of software reliability plays a crucial role in software development process. Software fault is an error situation of the software system that is caused by explicit and potential violation of security policies at runtime because of wrong specification and inappropriate development of configuration . Analyzing and predicting defects are needed for three main purposes, firstly, for assessing project progress and plan defect detection activities for the project manager. Secondly, for evaluating product quality and finally for improving capability and assessing process performance for process management. In fault prediction, previous reported faulty data with the help of distinct metrics identify the fault-prone modules. Important information about location, number of faults and distribution of defects are extracted to improve test efficiency and software quality of the next version of the software. Two benefits of software fault prediction are improvement of the test process by focusing on fault-prone modules and by identification the refactoring candidates that are predicted as fault-prone. Numbers of different methods were used for software fault prediction such as genetic programming, decision trees, neural network, distinctive Naïve Bayes approaches, fuzzy logic and artificial immune system (AIS) algorithms. Almost all software fault prediction studies use metrics and faulty data of previous software release to build fault prediction models, which is called *supervised learning* approaches. Supervised machine learning classifiers consist of two phases: training and test phase; the result of training phase is a model that is applied to the testing data to do some prediction. There are some other methods like clustering, which could be used when there are no previous available data; these methods are known as *unsupervised learning* approaches. It should be mentioned that some researchers like Koksal et al used another classification for data mining methods, which are famous as descriptive and predicative.

One of the main challenges in this area is how to get the data. In some works like in, a specific company provides the data, so the results are not fully trustable. Before 2005, more than half of the researches have used non-public datasets; however after that, with the help of PROMISE repository, the usage of public datasets reached to half, because the results are more reliable and not specific to a particular company. According to, software fault predictions are categorized based on several criteria such as metrics, datasets and methods. According to the literatures, software fault prediction models are built based on different set of metrics; method-level and class-level are two of the most important ones. Method-level metrics are suitable for both procedural and object-oriented programming style whereas class-level metrics are extracted based on object-oriented notation. It should be mentioned that compared to the other metrics, the method-level metrics is still the most dominant metrics prediction, followed by class-level metrics in fault prediction research area and machine-learning algorithms. It has been for many years that researchers work on different types of algorithms based on machine learning, statistical methods and sometimes the combination of them. In this paper, the experiments have been done on four NASA datasets with different population size using two distinct feature selection techniques that are principal component analysis (PCA) and correlation-based feature selection (CFS). The predictability accuracy has been investigated in this paper based on two different method-level metrics, which are 21 and 37 static code attributes. The algorithms in this study are decision tree (C4.5), Genetic Algorithms, Naïve Bayes, Support Vector Machines, A Fuzzy Model whics uses Software Metrics. Although we calculated accuracy along with above metrics, it does not have any impact on the evaluation process. Figure 1 shows the research done in this study.

II. A GENETIC ALGORITHM BASED CLASSIFICATION APPROACH FOR FINDING FAULT PRONE CLASSES

A. Genetic Algorithm Based Classification Technique

A genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of Evolutionary Algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. This

Technique used the feature of random search. Random search feature selection searches the best possible solution over a range of data. Random features and input given produce good result. In the beginning start with a large "population" of randomly generated "attempted solutions" to a problem then repeatedly do the following: • Evaluate each of the attempted solutions • Keep a subset of these solutions (the "best" ones) • Use these solutions to generate a new population • Quit when you have a satisfactory solution (or you run out of time) With help of Genetic algorithm classification of the software components into faulty/fault-free systems is performed. The flowchart of the Genetic Algorithm based approach is shown in the following figure:

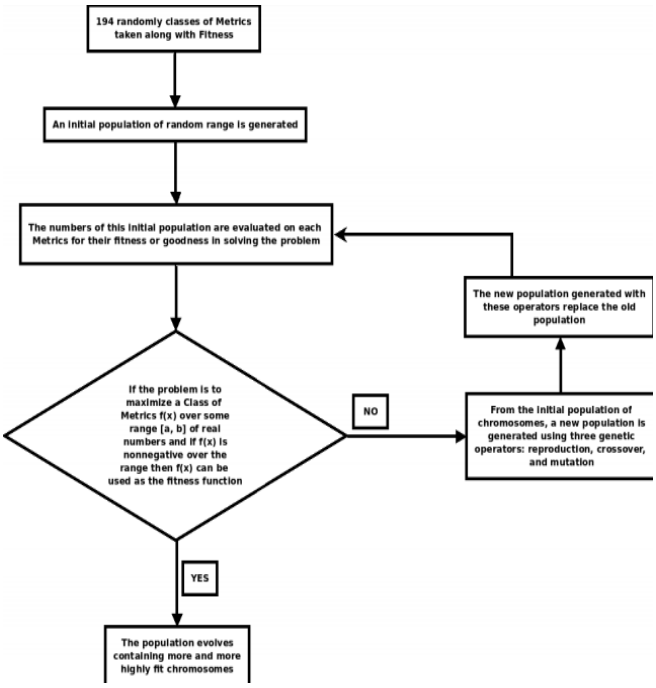


Fig. 1 Flowchart of use of GA

B. Empirical data collection

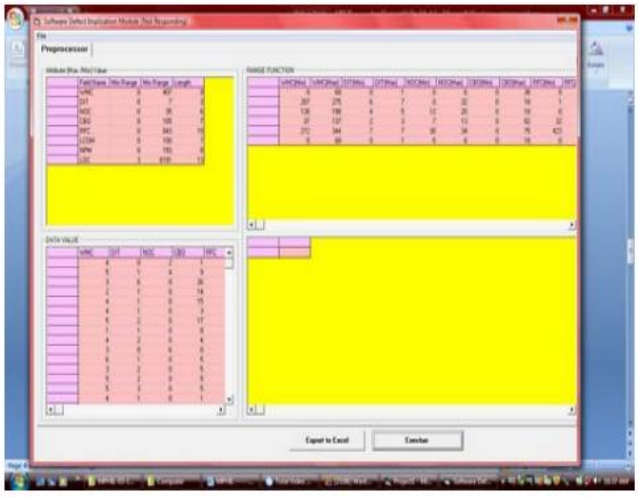
First of all, find the structural code and design attributes of software systems. Thereafter, select the suitable metric values as representation of statement. Next step is to analyze, refine metrics and normalize the metric values. We used JEdit open source software in this study. JEdit is a programmer's text editor developed using Java language. JEdit combines the functionality of Window, Unix, and MacOS text editors. It was released as free software and the source code is available World Academy of Science, Engineering and Technology 36 2009 486 on. JEdit includes 274 classes. The number of developers involved in this project was 144. The project was started in 1999. The number of bugs was computed using SVC repositories. The release point for the project was identified in 2002. The log data from that point to 2007 was collected. The header files in C++ were excluded in data collection. The word bug or fixed was counted.The following is the details of the metrics used in the classification process:

TABLE I METRIC SUIT USED IN THE STUDY

Metric	Definition
Coupling between Objects(CBO)	CBO for a class is count of the number of other classes to which it is coupled and vice versa
Lack of Cohesion (LCOM)	It measures the dissimilarity of methods in a class by looking at the instance variable or attributes used by methods
Number of Children (NOC)	The NOC is the number of immediate subclasses of a class in a hierarchy.
Depth of inheritance (DOI)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes
Weighted Methods per Class(WMC)	The WMC is a count of sum of Complexities of all methods in a class Consider a class K1, with Methods M1…… Mn that are defined in the class. Let. C_1, C_2....C_n be the complexity of the methods $WMC = \sum_{i=1}^{n} C_i$ If all the methods complexities are considered to be unity , then WMC = n the number of methods in the class.
Response for a class (RFC)	The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class. It is given by RFC= |RS|, where RS, the response set of the class $RS = M_i \cup_{all\ j} \{R_{ij}\}$
Number of Public Methods(NPM)	It is count of number of Public methods in a class
Lines of Code (LOC)	It is the count of lines in the text of the source code excluding comment lines

C. Result & discussion

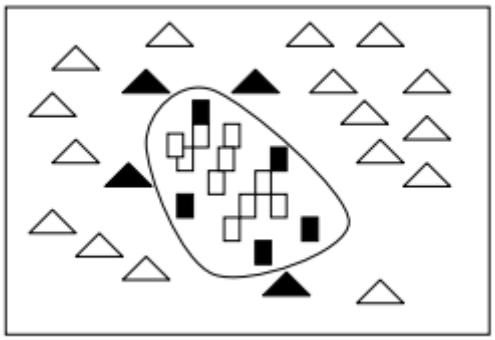
The rules for the classification of modules are generated with help of Genetic Algorithm based approach as shown in the flowchart of figure 1 and implemented in Visual Basic 6.0 environment. The snapshot of the GUI developed is shown in the following figure



III. Software Fault Proneness Prediction Using Support Vector Machines

A. *Model prediction using support vector machine (svm) method*

SVM are useful tools for performing data classification, and have been successfully used in applications such as face identification, medical diagnosis, text classification, pattern recognition, chinese character classification, and identification of organisms. SVM constructs an N-dimensional hyperplane that optimally separates the data set into two categories. The purpose of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that cases with one category of the dependent variable on one side of the plane and the cases with the other category on the other side of the plane. The support vectors are the vectors near the hyperplane. The SVM modeling finds the hyperpalne that is oriented so that the margin between the support vectors is maximized. When the points are separated by a nonlinear region, SVM handles this by using a kernel function inorder to map the data into a different space when a hyperplane can be used to do the separation. The recommended kernel function is the Radial basis Function (RBF). Thus, we used RBF function in SVM modeling to predict faulty classes in this study. The RBF kernel maps non-linearly data into a higher dimensional space, so it can handle non linear relationships between the dependent and the independent variables. Figure 1 shows the RBF kernel.



One category of the dependent variable is shown as rectangles and the Given a set of (xi, yi),…….,(xm, ym) and yi ∈{−1,+1} training samples. αi =(i=1,……,m) is a lagrangian multipliers. K(xi,yi) is called a kernel function and b is a bias. The discriminant function D of two class SVM is given below

$$D(x) = \sum_{i=1}^{m} y_i \alpha_i K(x_i, x) + b$$

Then an input pattern x is classified as

$$x = \begin{cases} +1 \text{ if } D(x) > 0 \\ -1 \text{ if } D(x) < 0 \end{cases}$$

The performance of the models predicted was evaluated using sensitivity, specificity, precision, completeness, and Area Under the Curve (AUC). Details on these measures can be found in [19]. The ROC curve, which is defined as a plot of sensitivity on the y-coordinate versus its 1-specificity on the x coordinate, is an effective method of evaluating the quality or performance of the predicted models.

B.Empirical Data Collection:

This study makes use of the public domain data set KC1 from the NASA Metrics Data Program [42]. The data in KC1 was collected from a storage management system for receiving/processing ground data, which was implemented in the C++ programming language. This system consists of 145 classes that comprise of 2107 methods, with 40K lines of code. KC1 provides both class-level and method-level static metrics. At the method level, 21 software product metrics based on product's complexity, size and vocabulary are given. Five types of defects such as the number of defects and density of defects are also given. At the class level, values of 10 metrics are computed including seven metrics given by Chidamber and Kemerer [15]. These seven OO metrics are taken in our study for analyses.

METRICS STUDIED

Metric	Definition
Coupling between Objects (CBO)	CBO for a class is count of the number of other classes to which it is coupled and vice versa.
Lack of Cohesion (LCOM)	For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged then subtracted from 100%.
Number of Children (NOC)	The NOC is the number of immediate subclasses of a class in a hierarchy.
Depth of Inheritance (DIT)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.
Weighted Methods per Class (WMC)	A count of methods implemented within a class.
Response for a Class (RFC)	A count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance.
Source Lines Of Code (SLOC)	It counts the lines of code.

C. Results of prediction model:

SLOC and CBO metrics have the highest values of the sensitivity and completeness. However, in case of NOC metric all the classes were predicted to be non faulty hence the results are not shown, as testing all the classes will be a high waste of the testing resources.

IV. Comparing The Effectiveness Of Machine Learning Algorithms For Defect PredictioN

A. Machine Learning Algorithms

Machine learning methods have been successfully applied for solving classification problems in many applications. The algorithms selected for classification on embedded dataset to compare the predictive effectiveness of J48

(Decision tree learner), OneR and Naïve Bayes (probabilistic learner). J48 is a JAVA implementation of Quinlan's C4.5 (version 8) algorithm. The J48 Decision tree classifier algorithm recursively splits a data set according to tests on attribute values in order to separate the possible predictions. The algorithm uses the greedy top-down construction technique to induce decision trees for classification. A decision-tree model is built by analyzing training data and the model is used to classify unseen data. J48 generates decision trees, the nodes of which evaluate the existence or significance of individual features. The decision trees are constructed in a top-down fashion by choosing the most appropriate attribute each time. An information theory measure is used to evaluate features, which provides an indication of the "classification power" of each feature. Once a feature is chosen, the training data are divided into subsets, corresponding to different values of the selected feature, and the process is repeated for each subset, until a large proportion of the instances in each subset belong to a single class. This algorithm is chosen to compare the accuracy rate with other algorithms.

The next scheme, OneR, produces very simple rules based on a single attribute. OneR is also useful in generating a baseline for classification performance. OneR algorithm builds prediction rules using one or more values from a single attribute. The OneR algorithm creates one rule for each attribute in the training data, and then selects the rule with the smallest error rate as its one rule. To create a rule for an attribute, the most frequent class for each attribute value must be determined. The most frequent class is simply the class that appears most often for that attribute value. A rule is simply a set of attribute values bound to their majority class. The error rate of a rule is the number of training data instances in which the class of an attribute value does not agree with the binding for that attribute value in the rule. OneR selects the rule with the lowest error rate. In the event that two or more rules have the same error rate, the rule is chosen at random. This algorithm is taken for comparing the strength of prediction with other algorithms, due to its simplicity and single attribute requirement.

The naive Bayes algorithms are based on theorem of Bayes posterior probability. Naive Bayes makes the assumption of class conditional independence i.e. there are no dependence relationship among the attributes. Learning a naive Bayes classifier is straightforward and involves estimating the probability of attribute values within each class from the training instances. Probabilities are estimated by counting the frequency of each discrete attribute values. For numeric attributes it is common practise to use the normal distribution. C4.5 is an algorithm that summarises the training data in the form of a decision tree. Learning a decision tree and rule is different process than learning a Naive Bayes model.

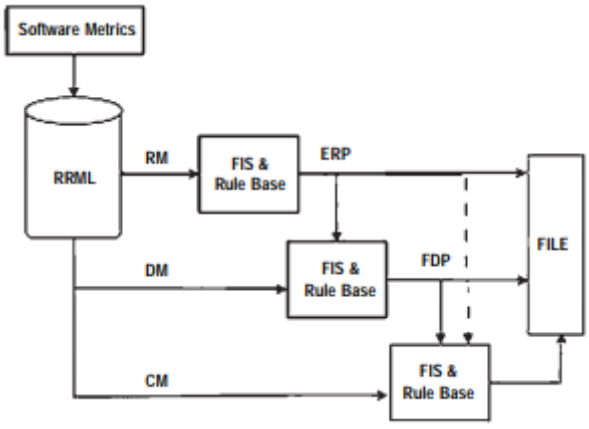
B. Results

The data sets used here is publicly available. The classification was done on freely distributed tool available online WEKA machine learning toolkit. In the experiment machine learning algorithms was used for performance analysis of two dataset having 121 modules and 29 metrics having 9 defective modules and other having 101 modules 29 static codes attributes and 15 defective modules. The selected features were assessed by running them through a 10-way cross validation over the J48, OneR and Naïve Bayes. The 10-fold cross-validation process splits the data into 10-equal disjoint parts and uses 9 of these parts for training the framework and 1 for testing. This is done 10 times, each time using a different part of data for testing. The training data are used initially to discretize the faults and then to train a classification algorithm. The learned model is then applied to the test data. The outputs of various machine learning algorithms are compared on the basis of confusion matrix. The performance of J48, OneR and Naïve Bayes for correctly classified instances are 90.086%, 89.2562% and 85.124% respectively for software having 121 modules.

V. A Fuzzy Model for Early Software Fault Prediction Using Process Maturity and Software Metrics

A. Proposed Model

Early faults prediction attracts both software professional as well as management as it provides an opportunity for the early identification of software quality, cost overrun and optimal development strategies. During the requirements, design or coding phase predicting the number of faults can lead to mitigating actions such as additional reviews and more extensive testing. The model considers two most significant factors, software metrics and process maturity together, for fault prediction. The model architecture is shown in the following figure. Software metrics can be classified in three categories: product metrics, process metrics, and



EARLY FAULT PREDICTION MODEL

metrics. Product metrics describe characteristics of the product such as size, complexity, design features, performance and quality level etc. Process metrics can be used to improve software development process and maintenance. Resources metrics describe the project

characteristics and execution. Approximately thirty software metrics exist, which can be associated with different phases of software development life cycle. Among these metrics some are significant predictor to reliability.

B. Implementation

The model is implemented in MATLAB utilizing fuzzy logic toolbox. The basic steps of the model are identification of reliability relevant input/output variables, development of fuzzy profile of these input/output variables, defining relationships between inputs and output variables and fault prediction at the end of each phase of software life cycle using fuzzy inference system (FIS). These basic steps can be grouped into three broad phases as follows: (1) Early information gathering phase, (2) Information processing phase, and (3) Fault prediction phase.

B-1. Information Gathering

Phase The quality of the fuzzy approximation depends mainly on the quality of information collected (subjective knowledge) and expert opinion. The information gathering phase is often considered the most vital step in developing a fuzzy logic system. The development of fuzzy profiles for identified input/output variables and fuzzy rules are assumed as building blocks of the fuzzy inference system and includes three steps

- Identify the Input and Output Variables
- Develop Fuzzy Profile of Identified Variables
- Develop Fuzzy Rule Base

B-2. Information Processing Phase

In this phase, the fuzzy system maps all inputs on to an output. This process of mapping inputs on to output is known as fuzzy inference process or fuzzy reasoning Basis for this mapping is the number of fuzzy IF-THEN rules, each of which describes the local behavior of the mapping. The Mamdani fuzzy inference system is considered here for all the information processing.

- ***4.2.1. Defuzzification***

  Defuzzification is the process of deriving a crisp value from a fuzzy set using any defuzzification methods such as Centroid, Bisector, Middle of maximum, Largest of maximum, and Smallest of maximum. The most commonly used method is the Centroid method, which returns the center of area under the curve, is used in here for defuzzification.

B-3. Fault Prediction Phase

The model resemble waterfall model. It divides the structure into three consecutive phase I, II, and III i.e. requirement phase, design phase, and coding phase respectively. Phase-I predicts the number of faults at the end of requirement phase using requirement metrics such as RCR, RIW, and PM. Phase-II predicts the number of faults at the end of design phase using design metrics such as DDD, FDN, and DC. Since most of the software faults traced back to requirements error, FRP is considered as input metric to this phase. Similarly at phase-III besides the coding metrics CC and CDD, FRP and FDP are also considered as input to predict the number of faults at the end of design phase. The Mamdani fuzzy inference system is considered here for fault prediction at each phase.

C. Results

In the proposed model, in order to analyze the impact of individual phase of software life cycle on the prediction of software faults, the values of the metrics from three different software projects are considered. Result of the best case and worst case input metrics applied to the proposed model state that the proposed model could be useful to predict the software faults that may range from 0 to 85. The number of fault at the end of requirement phase is 70 in worst case, 10 in average case and 1.33 in the best case. The number of fault at the end of design phase is 79.8 in worst case, 45 in average case, and 3 in the best case. Finally the number of fault at the end of coding phase is 84.7 in worst case, 56.7 in average case, and 5.67 in the best case.

VI. SUMMARY

Early fault prediction saves projects from budget overrun and risks. We discussed 4 approaches to fault prediction using machine learning algorithms on different reliability relevant software metrics and Capability Maturity Model (CMM) level.Results show that machine learning algorithms have good accuracy that can range from 80% to 90%. Machine Learning approaches can also help software maintenance developers to classifying software modules into faulty and non-faulty modules.

References:

[1] A Genetic Algorithm Based Classification Approach for Finding Fault Prone Classes by Parvinder S. Sandhu, Satish Kumar Dhiman, Anmol Goyal

[2] Software Fault Proneness Prediction Using Support Vector Machines by Yogesh Singh, Arvinder Kaur, Ruchika Malhotra

[3] Comparing The Effectiveness Of Machine Learning Algorithms For Defect Prediction by Pradeep Singh

[4] A Fuzzy Model for Early Software Fault Prediction Using Process Maturity and Software Metrics by Ajeet Kumar Pandey & N. K. Goyal.