Improve Software Quality using Defect Prediction Models

R. Sathya¹, Dr. P. Sudhakar²

¹,²Assistant Professor, Department of Computer Science & Engineering, Annamalai University, Annamalai Nagar, INDIA

ABSTRACT

In spite of meticulous planning, good documentation and proper process control during software development, occurrences of certain defects are unexpected. These are may lead to degradation of the quality which might be the underlying cause of failure. In today’s situation, it’s necessary to make conscious efforts to control and minimize defects in software engineering. However, these efforts cost, time and resources. This paper identifies important factors which in turn suggest the solutions to improve software quality and productivity. The paper also shows on how the various defect prediction models are implemented resulting in reduced magnitude of defects.

Keywords--- Defect Density, Defect Prediction, Machine Learning Algorithms, Software Defects, Software Quality

I. INTRODUCTION

Software metrics has been used to describe the complexity of the program and, to estimate software development time.

“How to predict the quality of software through software metrics, before it is being deployed” is a burning question, triggering the substantial research efforts to uncover an answer to this question. There are number of papers supporting statistical models and metrics which profess to answer the quality question. Typically, software metrics elucidate quantitative measurements of the software product or its specifications. Defects can be defined in a disparate way but are generally defined as deviation from specifications or ardent expectations which might lead to failures in procedure. Defect data analysis is of two types; Classification and prediction that can be used to extract models describing significant defect data classes or to identify future defects. Classification is used to identify categorical or discrete, and unordered labels, whereas prediction models predict continuous valued functions. Such analysis can help us for providing better understanding of the software defect data at large.

A software defect is an error, bug, mistake, failure, flaw or fault in a computer program or system that may generate an inaccurate or unexpected outcome software from behaving as intended. A project team always have high ambition to create a quality software product with zero or little defects. High risk components within the software project should be catches as soon as possible, in order to improve the software quality. Software defects always incurred cost in terms of quality and time. Moreover, identifying and solving defects is one of the most time consuming and expensive software processes. It is not practically possible to eliminate each and every defect but reducing the magnitude of defects.

In year 2008, SANS institute conducted a study to identify the most common and dangerous 25 software bugs or defects. Moreover, 25 organizations gave their contribution for the study. Commercials software organizations like Apple, Homeland Security, Breach Security, CERT, Microsoft, MITRE, Oracle, Red Hat, Aspect Security, and Tata; academic institutes like University of California, were among these organizations. These 25 security problems were classified into three domains [11] shown in figure 1.

Therefore, defect prediction is very important in the field of software quality and software reliability. Defect prediction is comparatively a novel research area of software quality engineering. By covering key predictors, type of data to be gathered as well as the role of defect prediction model in software quality; the interdependence between defects and predictor can be identified. This paper gives you much efforts to make understanding and future research chances about software defect prediction.

<table>
<thead>
<tr>
<th>Interactions</th>
<th>Resource Management</th>
<th>Defense Leaksages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor input validation</td>
<td>Unconstrained memory buffers</td>
<td>Inadequate authorization and access</td>
</tr>
<tr>
<td>Poor encoding of</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2016. Vandana Publications. All Rights Reserved.
Preemptive discovery of software defects in a software project gives power to managers to make appropriate decisions and plan limited number of project resources in a structured and systematic way. In general, we should concentrate on the following different aspects of the problem.

- Prevent the Defects,
- Detect the Defects,
- Correct the defects.

Since defect prediction is a relatively new area of research, in this paper we will discuss various prediction models which have been proposed. In the current prediction models, complexity and size metrics are used in order to take action in advance that any defects might occur during operation phase or testing phase of the project. In another model of defect prediction, reliability based models use the operational profile of a system to find out failure rate that the project will face. Almost in all projects, information collected in the testing and detection of defect is being analyzed help to predict defects for same types of projects. The multivariate model of defect prediction has been compelled as the model that can solve this issue but still no all included model has been uncovered as of now. But, it has become imperative to improve defect prediction techniques so that they can expect more defects at an early stage which leads to a delivery of good project.

### 1.1 A General Defect Prediction Process

To construct a prediction model, we must have defect and measurement data collected from actual software development efforts to use as the learning set. There exists conciliation between how well a model fits to its learning set and its prediction performance on additional data sets. Therefore, we should evaluate a performance of models by comparing the predicted defectiveness of the modules in a test set against their actual defectiveness [14].

Kim, S et al. [13] have described a common defect prediction process shown in the figure 2.

**Labeling:** Data’s are defected should be collected for training a prediction model. In this method, extraction of instances i.e. data items from software archives and labeling (TRUE or FALSE) is done.

**Extracting features and generating training sets:** It explains about extraction of features for prediction of the labels of instances. common features for defect prediction are complexity metrics, changes, structural dependencies and keywords. By combining features and labels of instances, we should produce a training set to be used by a machine learner to construct a prediction model.

**Constructing Prediction Models:** General machine learners such as Support Vector Machines (SVM) or Bayesian Network is also used to construct a prediction model by using a training set. The model can obtain a new instance and identify its label, i.e. TRUE or FALSE.

**Prediction and Assessment:** The evaluation of a prediction model requires a testing data set which is nearer to training set. The labels of instances in the testing set are identified and the prediction model is evaluated by comparing the prediction and real labels. 10-fold cross-validation is broadly used to separate the training and testing sets.
II. PROBLEM DEFINITION

As we have discussed upon earlier, defect prediction is essential in nature. Our aim is to predict defects without overrunning the estimated cost as well as without delaying the delivery of software in scheduled time. However, the important problem related to this is mainly the plethora of models which can be used for the same. All models of defect prediction have their own set of benefits and drawbacks which makes it hard to understand. In that situation, fault prediction model should be used and more importantly in what type of project. Since every project tends to be unique, this is hard from a decision making standpoint. However, we believe thorough model can examined that enable project managers to make a more informed decision.

In this paper, we will cover the popular models of defect prediction and evaluate the pros and cons of each model along with the situations where the models can be used. We will evaluate the models based a varied set of categories depending on the model being discussed. After evaluation, we will also include our personal observations and interpretations.

III. STUDY OF SOFTWARE DEFECT PREDICTION MODELS

3.1 Defect prediction Model using size and complexity metrics

From the familiar models of defect prediction, the approach that uses size and complexity metrics is well known. This model uses the program code as a basis for identification of defects. McCabe developed the concept of complexity model along with lines of code (LOC). Using regression equations, simple prediction metrics estimates can be obtained using a dependent variable (D) defined as the sum of defects found during testing and after 2 months’ post release. Famously, Akiyama made 4 equations. We have illustrated the equation that includes the LOC metric:

\[
\text{Defect (D)} = 4.86 + 0.018 \text{ Lines of Code (L)} \\
\text{D} = 4.2 + 0.0015 L_{4/3}
\]

Gaffney deduced above equation (1) into another prediction equation. He augmented that LOC was not language dependent owing to optimal size for individual modules with regards to defect density. The regression equation (2) is given above. The size and complexity models shows that defects are direct function of size and also shows that defects are occurred due to program complexity. This model avoids the underlying casual effects of programmers and designers. They are caused by the humans, who actually commence the defects, so any attribution for flawed code depends on individual(s) to certain extent. Capability of Poor design or difficulties in problem may result in high complex programs. Difficult problems might require complex solutions and naive programmers might create ‘spaghetti code’ [4].

3.2 Machine Learning Based Models

Machine learning (ML) algorithms has explained with great practical significance in solving a wide range of engineering problems which composed the prediction of software failure, error, and defects grows to be more complex. ML algorithms are very useful where problem areas are not well defined, human knowledge is limited and dynamic adaption for changing condition is needed, in order to develop efficient algorithms. Machine learning comprises of different types of learning such as artificial neural networks (ANN), decision trees (DT), Bayesian belief networks (BBN), reinforcement learning (RL), genetic algorithms (GA) and concept learning (CL), instance-based learning (IBL), genetic programming (GP), inductive logic programming (ILP), and analytical learning (AL) [1].

John, G., Langley., [2] developed RF method for prediction of faulty modules with NASA data sets. Prediction of software quality was introduced by Khoshgaftaar et al. [3] by using artificial neural network. In this model, they classified that modules as fault prone or non-fault prone, using large telecommunication software system. They compared their results with another non-parametric model achieved from discriminant method. Fenton et al. [4] give his suggestion to the use of Bayesian belief networks (BBN) for the prediction of faulty software modules. Elish et al. [5] recommended the use of support vector machines to predict the defected modules with context of NASA data sets. This model compares its prediction performance with other statistical and machine learning models. We have discussed few models in detail to enhances the understanding of Machine learning based prediction models.

3.2.1 The Probabilistic Model for Defect Prediction using Bayesian Belief Network

Fenton, Krause and Neil [4] proposed a probabilistic model for defect prediction. They recommended a model rather than a single factor (for e.g. size, or complexity, or testing metrics, or process quality data) model, by combining the different factors of casual evidence in order to successful defect prediction. The model uses Bayesian Belief Network (BBN) is the suitable example for representation of this evidence. The Bayesian approach causes statistical conclusion to be improved by expert judgment in those parts of a problem sphere where empirical data is scattered. Additionally, the causal or influence organization of the model better reflects the series of real world events and relations than any other practice.
BBN can be exploited to support effective decision making for SPI (Software Process Improvement), by executing the following steps.

Identification of variables (Hypothesis, Information or Mediating variables)

Defining the accurate relationships among variables

Achieve a probability distribution for each variable in the BBN

Fig 3. Bayesian Approach

A BBN represents the joint probability distribution for a set of variables. It can be achieved by defining Directed acyclic graph (DAG) and Conditional probability tables. A BBN can be employed to infer the probability distribution for a target variable (e.g., “Defects Detected”), which indicates the probability that the variable will obtain on each of its possible values (e.g., “very low”, “low”, “average”, “high”, or “very high” for the variable “Defects Detected”) given the observed values of the other variables [6,7]. N. Fenton, M. Neil and D. Marquez [12] reviewed the use of Bayesian networks to overcome disabilities of using BN’s for predicting software defects and software quality. This compelled modeller to predefine discretization intervals in advance and resulted in approximate predictions for large set of data. To improve this dynamic discretization algorithm was used. This algorithm exploits entropy error as the basis for approximation allowing more accuracy.

3.2.2 The Probabilistic Model for Defect Prediction using Fuzzy Logic Approach

This model is mainly based on the concept or reasoning and works on a value that is approximate in nature. It mainly works on conventional Boolean Logic where it returns only be True or False. In Fuzzy logic concept, the truth of any statement is degree and not an absolute number. Modeled on human intuition and behavior, the main thing of Fuzzy logic is that as supposed to the traditional yes – no answers, this model factors in the degree of truth and hence makes allocation for the more human like answers. Previously in this report, we have elaborated on why it is important to identify software quality issues at an early stage. Pandey A.K and Goyal N.K. [8] suggested the model of Fuzzy Logic and the software metrics and also the maturity of the project, the model can be constructed as follows:

It uses the inputs and puts them in a range system. After this, a set of rules is defined that dictates and influences how inputs will be utilized in getting the output as well as finding the definite value in the fuzzy set. The model has a set of metrics or reliability relevant metric (RRML) list which is made from the available software metrics. The metrics are pertinent to their respective phases in the software development life cycle.

Requirement Phase Metrics – In this model, it has three requirements metrics (RM). They are such as Requirements Change Request (RCR), Review, Inspection and Walk through (RIW), and Process Maturity (PM) as given as input to the requirements phase.

Design Phase Metrics – like as the above phase, this phase has three design metrics (DM) i.e. design defect density (DDD), fault days’ number (FDN), and data flow complexity (DC) have been considered as input.

Coding Phase Metrics – In this phase, there are two coding metrics (CM) such as code defect density (CDD) and cyclometric complexity (CC) have been taken as input at coding phase. The outputs of the model will be the number of faults at the end of Requirements Phase (FRP), number of Faults at the end of Design Phase (FDP), and number of Faults at the end of Coding Phase (FCP).

3.3 Defect Density Prediction Model

Defect density is used for the compare the defects in different software components. Also, used to identify high-risk components and associated resources. Moreover, it can also use for comparison among various software products in term of quality.

Defect density is a measure of the total confirmed defects divided by the software entity size is being
measured. The Number of Known Defects is defined as the count of total defects predicted against a particular software entity in certain time period. Most common defects are occurred in programs. Size attribute is like a normalizer that allows the comparisons between various software entities (i.e., modules, releases, products). Size is normally measured either in Lines of Code or Function Points [15].

Chulani S [9] discussed Constructive Quality Modeling for Defect Density Prediction (COQUALMO), a model used to predict quality. This software model focuses on the prediction of defect density and it is an estimation model. The COQUALMO model is generally applied to the two phases of the software lifecycle such as the activities of analysis and design. However, this COQUALMO model can also be applied to the later stages of the SDLC helping in refines the defect density estimate when a larger set of information is available. The COQUALMO model makes the project managers to get an estimate with relation to metrics like shipping time, payoffs for investing in quality strategies.

This model is mainly classified into two main phases namely,

- Defect Introduction model
- Defect removal model

Defect introduction model deals with that the defect can be introduced in any stage of the SDLC. classification is done by origin of the defects. Conceptually, this model can be through of being similar to a tank with specific pipes. These pipes relate to the origin of defects which in this case can be of three types, namely requirements, design and coding. The same has been illustrated in the given model.

As you can infer, the defects can be of different types. They are classified into Critical defects, low level and medium level defects. First, need most attention since they could make the system to crash or it may cause serious damage. The High level would be responsible for loss of the system’s critical functions without any measures for a workaround. Medium level is similar to the high level with the only difference being that a workaround solution will exist in this case.

**Defect Removal Model** is similar to the Defect Introduction model, this model is used to find defects in requirements, design and coding which are introduced into the product. This model aims to evaluate the removed defects. Classification of the defect removal activities are:

- Very low
- Low
- Nominal
- High
- Very High
- Extra High

The “very low” level is the least effective defect removal method and the extra high is the most effective defect removal method.

Dhiauddin, M, Suffian.M and Ibrahim.S [10] developed Six Sigma approach, which is a structured and systematic way to construct the mathematical model for prediction of functional defects in system testing. It focuses on V-Model software development process. Six Sigma methodology provides analysis from first phase to last phase. This prediction model is organized in to five phases; Define, Measure, Analyze, Design and Verify phases. These phases exhibit the progression and relationship between the outputs of each phase towards building the model. Define phase is used to create project definition and collecting primary requirements of the project. Measure phase is used for Measurement System Analysis (MSA) to validate the repetition and reproduction of defects. Analyze phase focuses on data collected earlier, is used to run regression analysis. Design phase is used to make extra refinement is carried out in the previous equation. Verify phase is the final phase, which is used to evaluate the reliability of the prediction model.

The Design for Six Sigma (DISS) methodology provides,

- A Control plan that guides on correct action when the genuine defects are discovered.
- The Six Sigma method builds a defect prediction models used for defect prediction.
- The Six Sigma provides the good opportunities to address the outline of issues.

**IV. CONCLUSION**

Software defect prediction is the process of find out the defective components in the software. Prediction is the task of predicting the continuous values for given input. we should try to minimize these defects to minimum count. Defect prediction models are important to achieve software quality. Size or complexity measures are simple regression models, which makes relationship between defects and program complexity. we have discussed, some classification techniques such as Bayesian belief networks, neural network can be adapted for prediction. Training a classifier or predictor is not enough, we would like an
estimate of how accurately the classifier can predict the future defects. We have observed various methods to construct more than one predictor and to estimate their accuracy. Defect prediction leads to reduced development time, cost, reduced rework effort, increased customer satisfaction and more reliable software. They presented probabilistic model based on Bayesian belief networks to overcome this problem.

We have discussed the various machine learning techniques for the software fault prediction problem. There are number of software defect prediction models available but in our paper, we got conclusion that these models heavily depend on the nature, volume of the defect data and accuracy of classifier and predictors. Most of the researches were carried out with the help of NASA defect data sets.

REFERENCES