1. INTRODUCTION

INTRODUCTION TO SOFTWARE DEFECT PREDICTION

There has been a huge spurt in the demand for software quality, especially over the last one or two decades. As a consequence, the issues related to testing are becoming increasingly critical. The ability to measure software defects can be extremely crucial for minimizing the cost and improving the overall effectiveness of the testing process. A majority of faults in a software system are found in a few of its own components.

Although there are several definitions for software quality, it is totally agreed that a project with numerous defects lacks quality in its software. Knowing the causes of possible defects as well as identifying general software process areas that may need attention starting from the initialization of a project could save money, time and effort. The possibility of early estimation of the probable faultiness of software could help on planning, controlling and executing software development activities. A low-cost method for defect analysis is learning from past mistakes to prevent future ones. Today, several data sets exist, which could be mined to gain useful knowledge with regard to the defects that transpired during the rapid growth in software development. The defects in software are due to various reasons. In the process of software development, testing of software is the main phase which reduces the defects in the software. If a developer or a tester can predict the software defects properly, it then reduces the cost, time and effort. In this thesis, we put forward a comparative analysis of software defect prediction based on classification rule mining. We propose an innovative scheme for this process and we choose different classification algorithms that helped in showing the comparison of predictions in software defects analysis. This evaluation analyzes the prediction performance of competing learning schemes with the given historical data set (NASA MDP Data Set). The result of this scheme evaluation shows that we have to choose different classifier rules for different data sets. Using this knowledge one should ideally be able to:

- Identify potential fault-prone software;
- Estimate the distinct number of faults; and
- Discover the possible causes of faults.
1.1 MOTIVATION

Different data mining methods have been proposed for defect analysis in the past, but only few of them managed to deal successfully with all of the above issues. Regression model estimations are difficult to interpret and also provide the exact number of faults. That is too risky, especially at the beginning of a project when too little information is available. On the other hand, classification models that predict possible faultiness can be specific, but they are not that useful to give any clues about the actual number of faults. Many researchers used numerous techniques with various datasets that predict faultiness, but there are multiple classification ruling algorithms that can be effective to predict faultiness. All the aforementioned issues motivated us in our research in these areas of software fault/defect prediction.

1.2 PROBLEM STATEMENT

The goal of this research is to help developers identify defects based on the existing software metrics using data mining techniques and thereby improve software quality, which ultimately leads to reducing the cost of software development both in the developing and maintenance phases. The focus is identifying defective modules so that the scope of software that needs to be examined for defects could be prioritized. This allows the developer to run test cases in the predicted modules. The proposed methodology helps in identifying modules that require immediate attention and the reliability of the software is vastly improved as the high priority defects are handled first. The goals of this research in particular are:

i) To propose a data mining approach to the software defect prediction process based on software metrics.

ii) To present architecture of a framework for software defect prediction process model based on the available metrics.

The overall goal is to propose a novel method for software defect identification using machine learning and various classification methods. The data set of software metrics used for this research is acquired from NASA’s Metrics Data Program (MDP). The purpose is to improve the classification accuracy of the data mining algorithm. To do this existing classification algorithms are evaluated initially using different Datasets.
1.3 OBJECTIVE

Keeping the research indications in view, it is realized that there exists enough scope to improve the software defect prediction. The objectives are confined to the following:

i) To utilize novel data set filtering mechanism for effective noise remove

ii) To utilize novel classification algorithm for better prediction

iii) To use better evaluation measurement parameter to get better result

iv) To decrease the software development cost, time and effort

1.4 RELATED WORKS

1.4.1 Regression via Classification

Bibi. Tsoumakas and Stamelos Vlahavas (2006) applied a machine learning approach to the problem of estimating the number of defects called Regression via Classification (RvC). Basili V.R (1996) applied the whole process of RvC that comprised two important stages:

i) The discretisation of the numeric target variable in order to learn a classification model, and

ii) The reverse process of transforming the class output of the model into a numeric prediction.

Static Code Attribute

Greenwald and Frank (2009) compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict software components containing defects. In order to do this, they used the NASA MDP repository which, at the time of their research contained 10 separate data sets.

Artificial Neural Networks

Iker Gondra and Bird (2011) used a machine learning method for defect prediction. They used artificial neural network as a machine learner.
1.4.2 Embedded Software Defect Prediction

Oral and Bener (2007) used layer perception for embedded software defect prediction. Chidamber (1994) implemented Multilayer Perception (MLP), Nave Bayes, VFI (Voting Feature Intervals) for embedded software defect prediction. They used only seven data sets for evaluation.

1.4.3 Association Rule

Baojun, Karel and Bacchelli (2010) used classification based association rule named CBA2 for software defect prediction. In this research they used association rule for classification and they compared it with other classification rules such as C4.5 and Ripper.

Defect-Proneness Prediction Framework

Jia, Ying and Liu (2011) proposed a general framework for software defect prone prediction. In this research they used M*N cross validation with the dataset (NASA, Softlab Dataset) for learning process and they used three classification algorithms (Naive baysed, OneR, J48) and they also compared them with MGFC framework of Bird (2009).

Chen, Sen and Du Ge (2010) developed a probabilistic model for software defect prediction for measuring accuracy of software. S. D. Conte (1986) proposed software defect prediction using data mining. In this research, they used probabilistic relational model and Baysean Network.

Approach for Software Defect Prediction

Software Defect (bug) Prediction is one of the most active research areas in software engineering according to M. D’Ambros (2010). He proposed defect prediction models that can provide the list of bug-prone software artefacts so that the quality assurance teams can effectively allocate limited resources for testing and investigating software products. We first introduce the common software defect prediction process and several research streams in defect prediction as different evaluation measures for defect prediction are used across the literature. Before building defect prediction models, some studies applied pre-processing techniques to improve prediction performance.
We briefly investigated the processing techniques used in the literature. We discussed about applications using defect prediction results and emerging topics. Finally, we concluded this survey by raising challenging issues for defect prediction.

1.5 OVERVIEW OF SOFTWARE DEFECT PREDICTION

A. Software defect prediction process

Figure 1.1 shows the common process of software defect prediction based on machine learning models. Most software defect prediction studies have utilized machine learning techniques suggested by A. Bacchelli & C. Bird (2011). The first step to build a prediction model is to generate instances from software archives such as version control systems, issue tracking systems, e-mail archives, and so on. Each instance can represent a system, a software component (or package), a source code file, a class, a function (or method), and/or a code change according to prediction granularity.

An instance has several metrics (or features) extracted from the software archives and is labelled with buggy/clean or the number of bugs. For example, in Figure 1.1, instances generated from software archives are labelled with ‘B’ (buggy), ‘C’ (clean), or the number of bugs. After generating instances with metrics and labels, we can apply pre-processing techniques, which are common in machine learning. Pre-processing techniques used in defect prediction studies include feature selection, data normalization, and noise reduction suggested by S. Kim & T. Menzies (2007). J. Nam and S. Shivaji (2013) proposed pre-processing is an optional step, but pre-processing techniques were not applied in all defect prediction studies.

With the final set of training instances, we can train a prediction model as shown in Figure 1.1. The prediction model can predict whether a new instance has a bug or not.
The prediction for bug-proneness (buggy or clean) of an instance stands for binary classification, while that for the number of bugs in an instance stands for regression.

B. Brief history of Software Defect Prediction Studies

Figure 1.2 shows the history of defect prediction studies in the last 50 years. The first study estimating the number of defects was conducted by Akiyama (1971). Based on the assumption that complex source code could cause defects, Akiyama built a simple model using lines of code (LOC) since LOC might represent the complexity of software systems. However, LOC is too simple metric to show the complexity of systems. In this reason, MaCabe and Halstead (1976), proposed the cyclomatic complexity metric and Halstead complexity metrics in 1976 and 1977 respectively. These metrics were very popular to build models for estimating defects in 1970s and the early of 1980s by N. Fenton (1999). Having said that though, the models studied in that period were not actually prediction models but just fitting models that investigated the correlation between metrics and the number of defects N. Fenton (1999). These models were not validated on new software modules. To resolve this limitation of previous studies, Shen et al. (1985) built a linear regression model and tested the model on the new program modules. However, Munson et al. (1992) claimed that the state of the regression techniques at that time were not precise and proposed a classification model that classifies modules into two groups, high risk and low risk. The classification model actually achieved 92% of accuracy on their subject system. However,
Munson et al. (1992)’s study still have several limitations such as no metrics for object-oriented (OO) systems and few resources to extract development process data as Shen et al. pointed out at that time that it was not possible to collect error fix information informally, as conducted by individual developers in unit testing phases.

In terms of object-oriented systems, Chidamber and Kemerer (1994) proposed several object-oriented metrics in software defect prediction and was used by Basili et al. (1996) to predict defects in object-oriented system. In 1990, version control systems were getting popular, and development history was accumulated into software repositories so that various process metrics were proposed from the middle of 2000.

In the beginning several limitations existed for defect prediction. The first limitation was the prediction model could be useful only after the product release for the purpose of quality assurance. However, it could be more helpful if we can predict defects whenever we change the source code. To make this possible, Mockus et al. (2000) proposed a defect prediction model for changes. Recently, this kind of model is called as Just-In-Time (JIT) defect prediction models. JIT prediction models have been studied by other researchers as T. Fukushima (2014) in recent years.

The second limitation is that it was not possible or difficult to build a prediction model for new projects or projects having less historical data. As the use of process metrics was getting popular, this limitation became one of the most difficult problems in software defect prediction studies as T. Zimmermann et al. (2009). To resolve this issue, researchers such as S. Watanabe (2008) proposed various cross-project defect prediction models. In cross-project defect prediction, identifying cross-prediction was another issue and researchers like Zimmermann et al. (2009) and Helix et al. (2012) conducted the study on cross-prediction feasibility.

The third limitation was from the question, “Are the defect prediction models really helpful in industry?” In this direction, several studies as E. Engström & C. Lewis (2013) have been done with case studies and proposing practical applications.

There were several studies using the trends of information technology as well such as using social network analysis and/or network measures. New metrics were proposed by T. Zimmermann et al. (2008). Privacy issue of defect datasets was addressed by Peters et al. (2012). New concepts of prediction models were proposed such as personalized defect
prediction model by T. Jiang (2013) and universal model by F. Zhang (2014) recently. In the next subsection, we categorized these studies by related subtopics.

C. Categories of Software Defect Prediction Studies

Table 1.1 lists the representative studies in software defect prediction. Many research studies in the last decade have focused on proposing new metrics to build prediction models. Widely studied metrics are source code and process metrics from F. Rahman (2013). Source code metrics measure how complex is the source code. The main rationale behind the source code metrics is that source code with higher complexity can be more bug-prone. Process metrics are extracted from software archives such as version control systems and issue tracking systems that manage all development histories. Process metrics quantify many aspects of software development process such as changes of source code, ownership of source code files, developer interactions, etc. Usefulness of process metrics for defect prediction is proved in many studies such as A. E. Hassan (2009). As introduced in this section, most of the defect prediction studies are conducted based on statistical approach, i.e., machine learning. Prediction models learned by machine learning algorithms can predict either bug-proneness of source code (classification) or the number of defects in source code (regression).

Some recent research studies showed that how machine learning techniques such as active/semi-supervised learning can be used to improve prediction performance. M. Li (2012) proposed BugCache algorithm that utilizes locality information of previous defects and keeps a list of most bug-prone source code files or methods (from S. Kim (2007)). BugCache algorithm is a non-statistical model and different from the existing defect prediction approaches using machine learning techniques.

Researchers also focused on finer prediction granularity. Defect prediction models tried to identify defects in system, component/package, or file/class levels. Studies like H. Hata & S. Kim (2008) showed that there is a possibility to identify defects even in module/method and change. Finer granularity can help developers by narrowing the scope of source code review for quality assurance.

Developing pre-processing techniques for prediction models is also an important research branch in defect prediction studies. Before building a prediction model, we may apply the following techniques: feature selection by S. Shivaji (2013), normalization by T. Menzies
With the pre-processing techniques proposed, prediction performance could be improved in the related studies.

Researchers also have proposed approaches for cross-project defect prediction. Most representative studies described above have been conducted and verified under the within-prediction setting, i.e., prediction models were built and tested in the same project. However, it is difficult for new projects that do not have enough development historical information, to build prediction models. Representative approaches for cross defect prediction are metric compensation by S. Watanabe (2008), Nearest Neighbour (NN) Filter by B. Turhan (2009), Transfer Naive Bayes (TNB) by Y. Ma (2012) and J. Nam (2013). These approaches use a prediction model by selecting similar instances, transforming data values, or developing a new model (S. Watanabe (2008)).

Table 1.1 Representative Studies In Software Defect Prediction

<table>
<thead>
<tr>
<th>Type</th>
<th>Categories</th>
<th>Representatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within/ Cross</td>
<td>Metrics</td>
<td>Source code by T. Menzies (2007)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Change by R. Moser (2008)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entropy by A. E. Hassan (2009)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Popularity by A. Bacchelli (2010)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authorship by F. Rahman (2011)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ownership by C. Bird (2011)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIM by T. Lee (2011)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network measure by A. Meneely (2008), M. Pinzger (2008), and T. Zimmermann (2008)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Anti pattern by S. E. S. Taba (2013)</td>
</tr>
<tr>
<td></td>
<td>Algorithm/ Model</td>
<td>Classification, Regression, Active/Semi-supervised learning by M. Li(2012) and B. Turhan(2013)</td>
</tr>
<tr>
<td></td>
<td>Finer prediction granularity</td>
<td>Change classification by S. Kim (2008)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method-level prediction by H. Hata (2012)</td>
</tr>
<tr>
<td></td>
<td>Preprocessing</td>
<td>Feature selection/extraction by S. Shivaji (2013) and N. Nagappan (2005)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Noise handling by S. Kim (2011) and R.Wu (2011)</td>
</tr>
<tr>
<td>Cross</td>
<td>Transfer Learning</td>
<td>Metric compensation by S.Watanabe (2008)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NN Filter by B. Turhan (2009)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TNB by Y. Ma, (2012)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCA+ by J. Nam (2013)</td>
</tr>
<tr>
<td></td>
<td>Feasibility</td>
<td>Decision Tree by T. Zimmermann (2009)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H. Hata by (2012)</td>
</tr>
</tbody>
</table>
Another interesting topic in cross-project defect prediction is to investigate feasibility of cross-prediction. Many studies confirmed that cross-prediction is hard to achieve; only a few cross-prediction combinations work according to T. Zimmermann et al. (2009). Identifying cross-prediction feasibility will play a vital role for cross-project defect prediction. There are a couple of studies regarding cross prediction feasibility based on decision trees by T. Zimmermann et al. (2009) and H. Hata (2012). However their decision trees were verified only in specific software datasets and were not deeply investigated.

Another interesting topic in cross-project defect prediction is to investigate feasibility of cross-prediction. Many studies confirmed that cross-prediction is hard to achieve; only few cross-prediction combinations work (T. Zimmermann et al. (2009)). Identifying cross-prediction feasibility will play a vital role for cross-project defect prediction. There are a couple of studies regarding cross prediction feasibility based on decision trees by T. Zimmermann et al. (2009) and H. Hata (2012). However these decision trees were verified only with specific datasets and were not deeply investigated.

1.5.1 Defect Prediction as a Classification Problem

Software defect prediction can be viewed as a supervised binary classification problem E. Arisholm (2007). Software modules are represented with software metrics, and are labelled as either defective or non-defective. To learn about different defect predictors, data tables of historical examples are formed where one column has a Boolean value for "defects detected" (i.e., dependent variable) and the other columns describe software characteristics in terms of software metrics (i.e., independent variables).

Binary Classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Binary or binomial classification is the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or not. Data Classification is a two-step process. In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning form" a training set made up of database tuples and their associated class
labels. In the second step, the model is used for classification. Therefore, a test set is used, make up of test tuples and their associated class labels.

A classification rule A. Bacchelli (2010) takes the form $X \Rightarrow C$, where $X$ is a set of data items, and $C$ is the class (label) and a predetermined target. With such a rule, a transaction or data record $t$ in a given database could be classified into class $C$ if $t$ contains $X$.

**Binary Classification Algorithms**

**Bayesian classification**

The Naive Bayesian classifier is based on Bayes theorem with independent assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

**Algorithm**

Bayes theorem provides a way of calculating the posterior probability, $P(c \mid x)$, from $P(c)$, $P(x)$, and $P(x \mid c)$. Naive Bayes classifier assumes that the effect of the value of a predictor $(x)$ on a given class $(c)$ is independent of the values of other predictors. This assumption is called class conditional independence.

\[
P(c \mid x) = \frac{P(x \mid c) P(c)}{P(x)}
\]

\[
P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)
\]

*Figure 1.3 Bayes Theorem*
i) \( P(c \mid x) \) is the posterior probability of class (target) given predictor (attribute).

ii) \( P(c) \) is the prior probability of class.

iii) \( P(x \mid c) \) is the likelihood which is the probability of predictor given class.

iv) \( P(x) \) is the prior probability of predictor.

**Example**

The posterior probability can be calculated first by constructing a frequency table for each attribute against the target. Then, transforming the frequency tables to likelihood tables and finally the Naive Bayesian equation is used to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Rule-Based Classification**

Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form: IF condition THEN conclusion

**Example**

\[
P(x \mid c) = P(\text{Sunny} \mid \text{Yes}) = \frac{3}{9} = 0.33
\]

\[
P(x) = P(\text{Yes}) = \frac{5}{14} = 0.36
\]

\[
P(c) = P(\text{Yes}) - \frac{9}{14} = 0.64
\]

\[
P(c \mid x) = P(\text{Yes} \mid \text{Sunny}) = 0.33 \times 0.64 = 0.36 = 0.60
\]

*Figure 1.4 Baye's Theorem Example*

IF *age=young* AND *student=yes* THEN *buys computer=yes*

There are many rule-based classifier algorithms such as Decision Table, OneR, PART, JRip, and ZeroR which can be used for classification.
Logistic Regression

In statistics, logistic regression or logic regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable (a dependent variable that can take on a limited number of values, whose magnitudes are not meaningful but whose ordering of magnitudes may or may not be meaningful) based on one or more predictor variables.

An explanation of logistic regression begins with an explanation of the logistic function, which always takes on values between zero and one:

\[ f(t) = \frac{1}{1 + e^t} \]

Decision Tree Classification

Decision tree induction is the learning of decision trees from class-labelled training tuples. A decision tree is a flow chart like tree structure, where each internal nodes (non-leaf node) denotes a test on an attribute, each branch represents an outcome of the test, each leaf node (or internal node) holds a class label. The topmost node in a tree is the root node. Figure 1.5 shows an example of a decision tree.

![Figure 1.5: Example Of Decision Tree](image)

There are many algorithms developed using decision tree for classification with some differences. Some of them like BF Tree, C4.8/J48, J48Graft, and Simple Cart are very popular.
The understanding of the area from introduction paved a path for a detailed literature review to identify specific objectives of this research is given in Chapter 2. A brief outline of this is given below:

*Chapter 2* describes the Literature Review and the proposed research, consisting of studying various aspects of software defect prediction, classification etc., which are published by various researchers. This requires study of the broad categories of classification algorithms and the related works done in the past by detailing the benefits and detriments of these different approaches to identify the gaps. Based on the identified gaps current research objectives are formed.