Predicting Bug Category Based on Analysis of Software Repositories

Mostafa M. Ahmed, Abdel Rahman M. Hedar, and Hosny M. Ibrahim

Abstract—Defective software modules cause software failures, increase development and maintenance costs, and decrease customer satisfaction. Understanding the impact of defects on various business applications is an essential way to improve software quality. For instance, we would expect that functionality bugs are fixed faster than other types of bugs due to their critical nature. Prior researches have often treated all bugs as similar when studying various aspects of software quality which is not a good way. Identifying bug category before debugging will lead to better handling to select proper destination to fix it. In this paper, ways to automatically predicting different bug types will be presented by using natural language mining techniques such as K Nearest Neighbor and Naive Bayes. Evaluating prediction techniques will be based on precision and recall measures [1]. We achieved the following recall and precision respectively 91% and 75% for standard related issues, 79% and 75% for function related issues, 79% and 73 % for user interface related issues and 72% and 79% for logic related issues, 79% is the highest accuracy achieved with Naive Bayes classifier.

Keywords— Predicting, Bug, Category, Software Repositories.

I. INTRODUCTION

This recent years, there is an increasing attention of reporting bugs resulting from software applications, which are considered as an important and precious source for application’s memory. Past errors play an important role in future work in building new software applications. This happens through avoidance of falling in the same errors or estimate the appropriate time and selecting good developers needed to solve new coming issues. Also, studies [2] show that more than 90% of the software development cost is spent on maintenance and evolution activities.

Software applications errors are managed and maintained by bug repository or issue tracking system [3]. Issue tracking system often contains a knowledge base containing information on each defect such as: description of the problem, resolution to fix it which is called impact analysis, project name, founder role, phase detected and phase Injected.

Issue track system is an open a communication channel between different people such as end users, programmers and testers to find the suitable response about detected issues in soft-ware applications.

Software bugs results due to several reasons including lack of requirements understanding, lack of good design of the software application, as well as the difficulty of implementing applications and lack of sufficient experience to coding it. There are different types of defects can be found in projects. Some defects can lead to method failure; some other defects can be deferred or missed such as spelling mistake in error message [4].

Defects must be classified according to its impact on the functionality of the application. For each defect type a proper developer is assigned to fix this type and a time unit is identified to resolve issue within it for example, security issues which are complex need much time and more experienced developer to fix them than performance issues [20].

During the emergence of mistakes with end users which are sent via issue tracking systems; project manager determines coordinator for these mistakes. Coordinator is a person who is responsible for selecting appropriate programmer and designer to investigate the problem. Investigator that includes programmer and designer determines the solution for the problem and estimate time needed to fix it as shown in Fig. 1. Also coordinator has fair experience to determine the type of mistakes and sort them according to their importance.

In giant projects, there are a quite large number of receiving defects daily. Coordinator should focus on deter-mining the nature of each issue to identify the correct per-son or action which is not an easy process. Assuming the errors reception rate in the system is 20-30 issues per day and the average time needed to determine defect type is 10 minutes, then coordinator will typically take 3-5 hours to determine right type to select the proper destination for these defects.

This is from time side, but from the practical side, in order to identify the defect type, coordinator have to view source code, system design and unit test documents to decide which category of new coming issue. Coordinator should have also enough knowledge about business applied in project to judge the type of new coming issue. Deter-mining nature of defect is a manual process which applied by coordinator. A lot of time and efforts is consumed in classifying bug type.

This paper discusses ways to automatically analyze bugs, classify their types and find similar characteristics in old projects. This will be a good achievement to save efforts and find the fix for the software application issue. Section 6 details

Mostafa Mohamed M. Ahmed is in Department of Information Systems, Assiut University. Email: mostafa.kamel@compit.au.edu.eg
Abdel Rahman M. Hedar is in Department of Computer Science, Assiut University. Email: hedara@aun.edu.eg.
Hosny M. Ibrahim is in Department of Information Technology, Assiut University. Email: hibrahim@aun.edu.eg.
the results of our experiments. In Section 7, the researcher concludes the paper with a discussion of results.

Software bugs are expressed in issue tracking systems with free text or through natural language. There is a large number of researches in processing natural language as an important pivot to find hidden patterns from data stored in the software repositories using mining techniques [5].

Mining techniques through defect description and impact analysis will be the best choice to identify automatically bug category of new coming issue. Through building a model to find patterns in different defect category classes and assign appropriate class to new coming issue. There are known data classification techniques such as Naïve Bayes, Support Vector Machine (SVM), Decision Trees and Neural Networks [6].

In this paper, we propose a classification based approach for building bug type identification, and conduct experiments using Naïve Bayes and K nearest classifiers. The paper will discuss the following questions:

1. What is the best solution to predict defect type? Is there any related work to fix this issue?
2. How effective is our model at predicting defect category?
3. How many bug reports are needed to train our model to get better results?

The rest of the paper is organized as follows: Section 2 discusses related researches about predicting bugs features. Section 3 gives an overview of the bug investigation process and bug report contents. Section 4 presents our proposed classification based approach and the classification techniques which we used for building the classifier. Section 5 show pre-steps to precede the classifier include text mining preprocessing.

There are numbers of researches about predicting bugs from source code by analysis them. Cathrin [8] build classifier that automatically predicts fixing effort. They use training dataset from the JBoss project. Classifier depends upon KNN [9] approaches by using on line summary of bug report. They make a new evaluation method to measure accuracy of their prediction which is called average absolute residual.

II. RELATED WORKS

Evaluation method represents the difference between the predicted effort $\pi$ and actual effort $e_i$. The prediction lies with 50% of the actual effort value.

Shuji [10] build a model to detect defect correction effort based on extended association rule mining. They defined defect fixing effort as a variable and appropriate association rule mining to treat with such variables. Data used are supported from Japan Ministry of Economy, Trade and Industry (METI). They use support and confidence as evaluation factors. Their approach expressed results as a mean of correction effort based on development level. For example, defects detected in coding and unit levels will be easy to correct (7% of mean effort) when they are coming with validation of input data.

Lamkan and Demeyer [11] predict severity of bug report using classification model for serve and non serve issues. They used online summary field of bug report for prediction based on SVM, Naïve Bayes, Multinomial Nave Bayes and Nearest Neighbor Classifiers which make better performance in results. Results were evaluated using ROC (receiver operating characteristic) curve [12]. Performance of Multinomial Naïve Bayes was found to be better than that of other classification algorithms.

Emad and Walid [13] have developed an approach for predicting re-opened defects through Eclipse projects, their study depend upon some factors such as work habits dimension like: day which issue is closed, the bug report features dimension like: components, the bug correction dimension like: time needed to fix bug. They evaluate their model with 62.9% precision and 84.5% recall when predicting whether a bug will be re-opened.

Kanwal and Maqbool [14] built a text approach using SVMs to prioritize new coming issues. They used bug report summary to prioritizing bugs. Accuracy of the classifier increased when training features were combined. Rune-son [19] predicts duplicated bug reports by finding the similarity of bug reports using the SVM. Results show that 2/3 of the duplicates can be found using this technique.

Kim[15] analyzed bugs classes according to bug life time. He invented a model by sorting bugs with a shorter life time as a higher priority level. Anvik and Hiew [16] apply mining techniques on the bug report data to predict who should fix new coming bug. They used Support Vector Machines (SVM), Naïve Bayes and Decision Trees algorithms on bug data of Bugzilla [17], Eclipse [18] projects. SVM achieved best result with precision 70%.

Zaman and Adams [20] who have analyzed the features of different types of bugs such as security and performance bugs to get useful information for their behavior in terms of the bug fix time, the number of developers assigned and the number of files impacted. Their Results show that security bugs are more complex, required more developers with experience, and large...
number of files affected but took less fix time than performance and other bugs. Similarly, performance bugs need more experienced developers than the other bugs.

Another important research about bug type detection is developed by Gegick and Rotella [21] who proposed a text mining technique to determine security bug reports (SBR) from the set of undefined non-security bug reports (NSBR). A bug report’s summary and long description fields were used for training the model. The bug data of Cisco software project was used to train model. The classifier is evaluated using the precision, recall and accuracy rate measures. Classifier is able to predict with percentage (78%) of SBRs that have been manually labeled as NSBRs. The research works on one kind of issues that are security and compared to manual selection of security issues.

From above discussion we did not find any related re-search on predicting bug category except Gegick, [21]. They deal with security issues only that are selected manually and no analysis found in open source data sets like Bugzilla and Eclipse on defect category and impact analysis of bug reports. Bugzilla does not interest in adding impact analysis details on bug reports. Whereas Gegick’s research select Cisco bug repository to reach impact analysis and identify bug classes.

III. ISSUE TRACKING SYSTEMS

Issue tracking systems [22] is used to manage and maintain lists of bugs which are received from different actors in development life cycle. It works as a record for software application characteristics. It also opens connection channel between end users, developers, designers and testers. Each one discusses his point of view for the new coming issue and change the status of the bug [23]. Communications happen through different activities such as creating entirely new issues, reading existing issues, adding details to existing issues, or resolving an issue.

When a user of the tracking system makes a change, all related date will be recorded such as the action and who made it. Recording all change details facilitate maintaining a history of the actions taken.

Each user of the tracking system may have issues assigned to him. He is responsible to find proper resolution to fix issue assigned to him. The user may have the option of reassigning an issue to another user, if needed. The tracking system will authenticate its users before allowing access to the systems for security purpose. Tracking system often also contains a knowledge base containing information on each user, resolutions to common problems, and other such data.

Bug report represents major component for issue track-ing which is called also a ticket. Ticket is created from technical support team, development team or testing team as a result of an incident. By creating a ticket a notification came to project manager or coordinator. Ticket has some details about business scenario that causes un-expected behavior from software application. In next two sections, we will describe bug report contents and bug life cycle.

Fig. 2 Bug Report Structure

Each user of the tracking system may have issues assigned to him. He is responsible to find proper resolution to fix issue assigned to him. The user may have the option of reassigning an issue to another user, if needed. The tracking system will authenticate its users before allowing access to the systems for security purpose. Tracking system often also contains a knowledge base containing information on each user, resolutions to common problems, and other such data.

Bug report represents major component for issue track-ing which is called also a ticket. Ticket is created from technical support team, development team or testing team as a result of an incident. By creating a ticket a notification came to project manager or coordinator. Ticket has some details about business scenario that causes un-expected behavior from software application. In next two sections, we will describe bug report contents and bug life cycle.

A. Bug Report Contents

A major component of an issue tracking system is bug re-port, which has specific information about incident and a set of fields. Some fields describe incident in natural language without any rules and there are some other fields that have predefined values. For example, description, impact analysis and issue title which represents free text fields in addition to, other predefined fields such as severity, project, founder Role, phase detected and phase injected as shown in Fig 2.

Title of the incident has the summary in one line for the problem that faces the reporter.
Whereas description represents a full scenario to regenerates the issue. Severity is defined as how important this issue is affecting on the software application. Also, it has range of values such as critical / blocker, major, medium, minor and low. Founder role which represents who found this issue such as tester, end user, developer or analyst. Phase detected is defined as in which phase this issue is generated. The phases values depend upon organization development cycle that may include requirement, design, coding, function test and user acceptance test. Phase Injected represents in which phase issue come from. Finally, impact analysis which represents why this issue is generated and what are the changes to fix it.

Defect category represents how this issue affects on application and has range of values. Defect category may be function, standards, graphical user interface or logic related. Table 1 shows each defect categories and types that represent it.

Bug report may have screen shots about problem found in software application. It is recommended for the reporter of the problem to upload log file that record all steps proceeded by software application. Also comments about raised bug report are recorded to clear manipulation of report.

B. Bug Life Cycle

Issue tracking systems have different states for a bug which is tracked through status assigned to it. At the moment an issue report is submitted, it gets a unique identifier by which it can be referred to in further communication. Let us assume that someone has just raise an issue report into the bug database. While the issue is being processed, the report runs through a life cycle as given in Fig. 3.

The position in the life cycle is determined by the state of the issue report. Initially, every single issue report has a state of New Issue. It is then checked for validity and uniqueness by coordinator. If it passes these checks, the proper developer / designer will be assigned. Identifying nature of the issue is crucial in when selecting proper developer and finding the resolution of issue. Status of the report becomes Assigned. At this point, the issue report is also assigned a priority the higher the priority, the sooner it is going to be addressed.

The developer now works on the issue; the state of issue is then changed to Under Investigation. If developer finds a problem in source code, system design or application configuration and comes up with a resolution. The status will be Resolved, otherwise the issues will be closed. At this stage, developer records the defect category. As the problem is now fixed, two more steps remain: the testers must confirm the success of the fix (resulting in under testing state). If the tester found issue is fixed, report will go to state Closed. Otherwise issue will go back to Under Investigation state.

IV. PROPOSED SOLUTION

In this section we briefing describe data classification and the role it plays in our solution, then we will describe classification techniques such as Naive Bayes and K Nearest and how we apply those techniques to get the best results in predicting defect category.

<table>
<thead>
<tr>
<th>Defect Category</th>
<th>What Causes Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Related</td>
<td>Improper naming convention Incorrect spelling/grammar Nonstandard Document format Absence of code comments Hard coding of code Incorrect description within Documents/ checklists</td>
</tr>
<tr>
<td>Logic Related</td>
<td>Incorrect processing steps in coding Abnormal ending of a program (Core dump) Incomplete processing steps in Coding Improper operations / shutdown due to DB Performance problem</td>
</tr>
<tr>
<td>Function Related</td>
<td>Missing functionality in design/requirements Incorrect naming or Missing of item in DB Requirement mismatch customer Needs Wrong functionality in design Extra undesired functionality in design/requirements Incorrect processing steps in the design/requirements</td>
</tr>
<tr>
<td>User Interface Related</td>
<td>Improper field length Wrong data validation Improper field format Improper size of page Form item is missing menu, buttons Improper display (of form, report, web page, etc..) Incorrect or missing error message</td>
</tr>
</tbody>
</table>

Fig. 3 Bug Life Cycle

http://dx.doi.org/10.15242/IIE.E0314580
A. Data Classification

Data surround us through scientific, demographic, financial and marketing fields. People have no time to look at these data. Human attention has become the important resource. Researches invented techniques to automatically summarize data, and discover hidden patterns from large amount of data to transform them into useful information which called mining techniques. There are different data mining techniques for recognizing patterns from the data, e.g., classification, association rule mining and clustering [24]. Data mining classification techniques classify the data according to some predefined categorical labels.

Classification is the process of building a model by learning from a dataset. Classification is a two-step process, learning and classification. In the first step a classifier model is built by determining the characteristics of each class from the given training dataset which consists of training instances with associated class labels [25]. For example, suppose X(x1, x2, x3,...,xn) is a training instance where x1, x2, x3,...,xn represent the features (attributes) and n is the number of features.

Each training feature provides a piece of information to the classifier that helps in determining the characteristics of the class. For each XI there is a special attribute (class label) which represents its class YI. This step can be viewed as learning of a function y = f(x) where y is the predicted class label and f(x) may be some rules or mathematical formulae. In the second step, the function is used to predict the class label y for new instances. In case of a rules-based classifier, rules are used to characterize the new instance in an appropriate class. In case of mathematical formula, feature values are plugged into the equation to find its class label.

Accuracy of the classifier is the percentage of test samples correctly classified by the learned classifier model. The dataset that is used for testing or validating the classifier is unseen data (not used for training). In the validation process, we know the actual class labels of the test data but the classifier is not aware of them. Although a number of classification algorithms are available e.g., Naive Bayes, Support Vector Machine (SVM), Decision Trees and Neural Networks. In the following subsections we describe Naive Bayes and K-Nearest Neighbors. These two classifiers have been used by various researchers for text classification and have shown promising results as found in [26]-[23].

B. Naive Bayes Classifier

The Naive Bayes algorithm classifies a new instance by calculating its probability for a particular class using the Bayes rule of conditional probability [27]. The probability of a new instance is calculated for each class and the class with the greatest probability is assigned to the new instance.

Suppose you have a collection of documents D1 ... Dn labeled as belonging to categories C1 ... Ck. A new document D is given for you to categorize. In a probabilistic approach, we are looking for the category C such that Prob(C|D) is maximal which mean that the probability that D is belong to class C. So how do we evaluate Prob(C|D)? The multinomial Naive Bayes method proceeds as given in equation [1]

\[ P(\text{rob}(C|D)) = \frac{\text{Prob}(D|C) \times \text{Prob}(C)}{\text{Prob}(D)} \]  

Let T1, T2 ... Tm be the sequence of lexical terms in D. We will assume that the occurrence of a term T in the Ith place of D depends only on the category C and, given C, is conditionally independent of all the other terms in D and of the position I. Calculation will be as given in equation [2] where Prob(Ti (C) means the probability that a randomly chosen word token in C is the word Ti and m is the number of words in D.

\[ \text{Prob}(D|C) = \prod_{i=1}^{m} \text{Prob}(T_i |C) \]  

We estimate Prob(Ti (C) and Prob(C) using the training set and calculated as the relative frequency of Ti in documents of category C as given in equation [3].

\[ \text{Prob}(T_i |C) = \frac{\text{number of occurrences of } T_i \text{ in } C}{\text{total number of words in } C} \]  

Prob(C) is estimated as the fraction of documents in the training set that are of category C. while Prob(D) is independent of the category C. Since we are given D and are just trying to compare different categories, we need not to calculate Prob(D). (This is known as a normalizing factor) its function is to ensure that the probabilities add up to 1. We can now calculate the product for each category Ci as given in equation [2] and choose the category that maximizes this.

Laplacian Correction

If document contains a term T that does not occur in any of the documents in category C, then Prob(T |C) will be estimated as 0. Then the product found in equation 2 will be equal to 0, no matter how much other evidence is favoring Ci.

The ususal solution is to do a Laplacian correction by upping all the counts by i to 1. That is, for multinomial Naive Bayes, let CT be the total number of occurrences of word T in documents of category C. Let V be the number of different words in D. Let |C| be the sum of the lengths of all the documents in the category C. Estimating Prob(T |C) as given in equation [4].

\[ \text{Prob}(T |C) = \frac{1 + CT}{|C| + V} \]  

Note that the sum over T of Prob(T |D) is equal to 1, as it should be. The Laplacian correction can give strongly counter-intuitive results than normal algorithm.

C. Nearest Neighbors Classifier

Nearest Neighbor classifiers rely on route learning. At training time, a Nearest Neighbor classifier memorizes all the documents in the training set and their associated features to classify a new document D. The classifier selects the k documents in the training set that are closest to D, then picks one or more categories to assign to D, based on the categories assigned to the selected k documents.
We have to transform document D into a representation suitable for the learning algorithm. The most commonly used document representation is called vector space model [28]. In this model, each document is represented by a vector of words. To define a K-NN (k-Nearest Neighbors) classifier, we first need to define the distance metric used to measure how close two documents are to each other.

We could use the Euclidean distance [29] between documents in the vector space. A word-by-document matrix A is used for a collection of bugs, where each entry represents the occurrence of a word in a document, i.e., \( A = a_{ij} \), where \( a_{ij} \) is the weight of word \( i \) in document \( j \).

There are several ways of determining the weight. Let \( f_{ij} \) be the frequency of word \( i \) in document \( j \), \( N \) the number of documents in the collection, \( M \) the number of distinct words in the collection, and \( n_i \) the total number of times word \( i \) occurs in the whole collection. The simplest approach is Boolean weighting, which sets the weight \( a_{ij} \) to 1 if the word occurs in the document and 0 otherwise. Another simple approach uses the frequency of the word in the document, i.e., \( a_{ij} = f_{ij} \). A more common weighting approach is called TF.IDF (term frequency - inverse document frequency) weighting as given in equation [5].

\[
a_{ij} = f_{ij} \times \log \frac{N}{n_i} \tag{5}
\]

A slight variation of the TF.IDF weighting, which takes into account that documents, may be of different lengths, as given in equation [6].

\[
a_{ij} = \frac{f_{ij}}{\sum_{i=1}^{M} f_{ij}^2} \times \log \frac{N}{n_i} \tag{6}
\]

To classify a class-unknown document \( X \), the K-Nearest Neighbor classifier algorithm ranks the document’s neighbors among the training document vectors, and uses the class labels of the K most similar neighbors to predict the class of the new document. The classes of these neighbors are weighted using the similarity of each neighbor to \( X \), where similarity is measured by Euclidean Distance or the cosine value between two document vectors. The cosine similarity is given in equation[7].

\[
sim(X, D_j) = \frac{\sum_{t_i \in (X \cap D_j)} x_i \times d_{ij}}{||X||_2 \times ||D_j||_2} \tag{7}
\]

Where \( X \) is the test document, represented as a vector. \( D_j \) is the training Document; \( t_i \) is a word shared by \( X \) and \( D_j \). \( x_i \) is the weight of word \( t_i \) in \( X \). \( d_{ij} \) is the weight of word \( t_i \) in document \( D_j \) is the norm of \( X \), and \( ||X||_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots} \) the norm of \( X \).

By adding a little modification on algorithm to have better result. Instead of select first nearest neighbor we do the following procedures:

1. Getting the average distances of K nearest neighbors.
2. Fetch all neighbors that are greater than average value.
3. Calculate the count of each class and the largest is taken.

As given in equation [8] represent above steps. After repeating experiment on different numbers for K. The best K that has been chosen is 30.

\[
\text{AverageDistance} = \frac{\sum_{i=1}^{K} \text{sim}(X, D_i)}{K} \tag{8}
\]

V. EXPERIMENT SETUP

A. Selected Issue Tracking System

We use Fast Issue Track (FIT 4) as a bug repository for Bug reports, which focus on Telecom and Banking bugs, it have many projects and over 240,000 issues and we use telecom products, we selected the platform product for our experiment as it contains larger number of bug reports according to bug category as compared to other products. The number of bug reports of training dataset is given in TableII. And the definition for each category and the causes is found in section 3.

<table>
<thead>
<tr>
<th>Defect Category</th>
<th>Number of issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Related</td>
<td>1100</td>
</tr>
<tr>
<td>Standard Related</td>
<td>720</td>
</tr>
<tr>
<td>Logic Related</td>
<td>1455</td>
</tr>
<tr>
<td>User Interface Related</td>
<td>975</td>
</tr>
</tbody>
</table>

B. Text Mining Preprocessing

Text attributes extracted from the bug reports are bug title and description. Text attributes contain a number of words that are not meaningful. So we applied a standard text categorization approach to transform the text data into a meaningful representation. First, the whole text was converted into words by removing punctuation, brackets and special symbols (e.g., @, $, %). From the list of words, stop words (e.g., is, am, I, he), common words (e.g., actually, because, everywhere) and non-alphabetic words (e.g., numbers, dates, currencies) were removed be-cause these are unimportant and provide little information about the problem described in the bug report.

Stemming is applied on the text to convert a word into its ground meaning. For example, “experimental” and “experiments” are converted to experiment. Verbs are also converted into their original form, e.g., “was” and “being” are converted to “be”. After finding the ground form of each word, the number of occurrences of each word in the bug report is calculated. Due to stemming the text, a word with different grammatical structure is considered as one word. So a word is represented in a vector form having two dimensions: a word...
and its frequency. All the categorical and text features are converted into numeric representation because K Nearest takes only numeric features within training and testing.

C. Evaluation Criteria

In order to evaluate the performance of the classifiers Nave Bayes and K Nearest we use precision and recall that are useful technique for organizing classifiers and visualizing their performance.

By considering classification techniques using four known categories. Formally, each instance I is mapped to one element of the set Function related, Logic related, Standard related and User interface related of class labels. A classification model is mapping from instances to predicted classes.

Given a classifier and an instance, the possible outcomes for the classifier will be true prediction in case of the prediction matches with predefined defect category. Otherwise it will be false one. There are some expressions to ensure the validity of the evaluation method.

TABLE 3
CONFUSION MATRIX

<table>
<thead>
<tr>
<th>Predicted Classes</th>
<th>True Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Logic</td>
</tr>
<tr>
<td>Function</td>
<td>TP Func</td>
</tr>
<tr>
<td>Logic</td>
<td>FP Func</td>
</tr>
<tr>
<td>Stand.</td>
<td>FP Func</td>
</tr>
<tr>
<td>GUI</td>
<td>FP Func</td>
</tr>
</tbody>
</table>

First, precision [1] which is defined as the percentage of bug reports correctly predicted. We consider precision thus for each defect category separately. When DC is the defect category, we define precision more formally as given in equation [9].

\[\text{Precision}(DC) = \frac{\#\text{bugs correctly predicted as DC}}{\#\text{bugs predicted as DC}}\]  

Where Recall is defined as the percentage of all bug reports that actually predicted as DC. Here, we also consider recall for each precision separately. We define recall as given in equation [10].

\[\text{Recall}(DC) = \frac{\#\text{bugs correctly predicted as DC}}{\#\text{bugs of Defect Category as DC}}\]  

And FP Rate (also called false alarm rate) of the classifier which defined in equation 11.

\[\text{FP rate}(DC) = \frac{\#\text{bugs incorrectly classified DC}}{\#\text{bugs predicted as DC}}\]  

We calculate both precision and recall using a confusion matrix, sketched in Table 3. This matrix represents all possible outcomes when making predictions of the defect category.

According to the matrix found in table 3 we calculate the precision, recall, FP and TP rates for each Defect Category (DC) as follow:

\[\text{Precision}[DC] = \frac{\sum (TP_{DC})}{\#\text{samples predicted as DC}}\]  

\[\text{Recall}[DC] = \frac{\sum (TP_{DC})}{\#\text{Actual samples in DC}}\]  

\[\text{FP Rate}[DC] = \frac{\sum (FP_{DC})}{\#\text{Actual samples in DC}}\]  

\[\text{Accuracy} = \frac{\sum \text{TP for all Categories}}{\#\text{Total Dataset}}\]

D. Training And Testing

We use K-Fold Cross [31] validation approach in testing data set. This approach first splits collection of bug reports into separated training and testing sets, then trains the classifier using the training set. Finally, the classifier is executed on the evaluation set and accuracy results are calculated. These steps are executed K times. For example in the case of 10-fold cross validation, the available bug reports are split randomly into 10 subsets. The classifier is trained using only 9 of the subsets and the classifier is executed on the remaining subset.
VI. EXPERIMENTAL RESULTS

In this section, we present the results of our experiment using K Nearest Neighbors and Naive Bayes classifiers. We try to answer the challenge questions presented in section 1. We also evaluate different categories and training dataset sizes. Accuracy of the classifier is measured using precision and recall approach as found in previous section. Accurate classifier will be chosen based on comparison between each classifier.

As found in Table IV the precision and recall measures for each of category for K Nearest Neighbor and Naive Bayes classifier. Results for Naive Bayes are presented by precision and recall respectively from 0.72 to 0.82 and from 0.78 to 0.79 for function related issues. However K Nearest achieve from 0.79 to 0.88 and 0.66 to 0.70.

For standards related issues precision and recall are vary from 0.7 to 0.74 and from 0.7 to 0.91 in Naive Bayes; whereas in K Nearest precision vary from 0.5 to 0.68 and recall from 0.51 to 0.62.

For Logic related issues precision vary from 0.77 to .81 and recall from 0.78 to 0.79 in K Naive Bayes; whereas in K Nearest precision vary from 0.55 to 0.87 and recall vary from 0.55 to 0.65. However precision vary from .5 to .68 and recall from 0.51 to 0.62.

For GUI related issues precision vary from 0.68 to 0.77 and recall from 0.72 to 0.81 in K Naive Bayes; whereas in K Nearest precision vary from 0.51 to 0.83 and recall vary from 0.52 to 0.71.

In Fig. 10 graphs, the FP rate measure for Naive Bayes which range from 0.19 to 0.38 whereas K Nearest varies from 0.3 to 0.5 as shown in Fig. 9. In Fig. 8 draw the accuracy measure for Naive Bayes that is vary from 0.75 to 0.79 whereas K Nearest vary from 0.51 to 0.88.

A. How Effective Is The Selected Model At Predicting Defect Category?

From the above result we found that Naive Bayes can get proper defect category with accuracy 75 % for training data set with 500 issue , accuracy of 79 % for training data set with 1000 issues, whereas get accuracy with 75 % with training dataset of 1500 issues and 76 % in case of training dataset of 2000 issues.
As result shows that Naive Bayes give better prediction than K Nearest when training data set size increased. However for each category we get precision and recall of each classifier. K Nearest get better results in case of training data set of size 500 i-su-sues in standards related issues, but for other categories in different training dataset size Naive Bayes wins.

B. How Many Bug Reports Are Needed To Train Our Model To Get Better Results?

We would like to know how many bug reports will need in order to obtain good and stable predictions. We ran a series of measurements, where gradually increase the size of the training set. The accuracy of the model increased with each increment of the training set until reaching to the maximum accuracy. The best accuracy occurred with training set of size 1000 issues. The accuracy reaches to 79% for Naive Bayes and after more enrichments the accuracy is down.

VII. CONCLUSION

Understanding nature of the bug is key solution for finding proper fix in suitable time. There is a need for a mechanism to support bug triaging by analysis of bug repositories. In this paper text mining techniques applied on bug repositories to identify the category of the defect after it is raised. The researcher uses Naive Bayes and K-Nearest classification algorithms and compare between them to find out which particular algorithm is best suited for classifying bug reports to proper defect category.

Evaluation of the classifier is measured by Precision and Recall technique. Naive Bayes get the best accuracy with different training set. Therefore we conclude that Naive Bayes is proper algorithm for the purpose of classifying bug reports.

This study tries to implement an automated classification model to detect bug category to better bug triaging process. By investigating related work on bug repository analysis, the researcher tries to contribute to improve reliability of bug triaging process through combining this study with other research.

Future work is aimed to find out more features that support our bug triaging process. Studying impact analysis section in bug reports will lead to better fix of different bugs for example, the most defective sections in business application takes more time to be tested and identifying testing effort needed to test such defective section is not an easy process. We can find appropriate testing effort for specific issue before testing process begins through studying Impact analysis section.

REFERENCES

### TABLE IV

**COMPARISON BETWEEN NAIVE BAYES AND K NEAREST ALGORITHMS**

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Size</th>
<th>Accuracy</th>
<th>Function</th>
<th>Logic</th>
<th>Stand</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>500</td>
<td>0.77</td>
<td>0.78</td>
<td>0.77</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.79</td>
<td>0.82</td>
<td>0.81</td>
<td>0.72</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>1500</td>
<td>0.75</td>
<td>0.75</td>
<td>0.79</td>
<td>0.68</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>0.76</td>
<td>0.72</td>
<td>0.79</td>
<td>0.69</td>
<td>0.70</td>
</tr>
<tr>
<td>K Nearest Neighbors</td>
<td>500</td>
<td>0.88</td>
<td>0.88</td>
<td>0.66</td>
<td>0.87</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.57</td>
<td>0.79</td>
<td>0.67</td>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>1500</td>
<td>0.54</td>
<td>0.79</td>
<td>0.7</td>
<td>0.7</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>0.51</td>
<td>0.79</td>
<td>0.67</td>
<td>0.53</td>
<td>0.55</td>
</tr>
</tbody>
</table>