## Background

## Literature Review

Classification of software bugs using bug attribute similarity (CLUBAS) is proposed by N. K. Nagwani et. al. [1]. The proposed work can be divided into five steps- data pre-processing, text clustering, frequent term calculations, taxonomic terms mapping techniques and performance evaluation. In the first step data are pre-processed by eliminating stop words and applying stemming over the textual bug attributes. In the second step text clusters are created using textual similarity between the attributes summary and description for each pair of bugs. In the third step cluster labels are generated calculating the frequent and meaningful terms from each cluster text data and assign them to that cluster. In the fourth step cluster labels are mapped against the bug taxonomic terms to identify appropriate categories of the clusters. In the fifth step performance is evaluated by calculating accuracy, precision, recall and F-measure.

The algorithm CLUBAS shows stability and performs better than Classification using Clustering (CC), Support Vector Machine (SVM) and J48 when this algorithm is implemented on the Android bug repository. It maintains the F-measure value more than 0.9 for each experiment using different number of samples. When the precision, recall and F-measures are important CLUBAS gives the better and stable results irrespective of number of samples and software bug repositories. But both Naïve Bayes (NB) and Naïve Bayes Multinomial (NBM) performs better in terms of accuracy than CLUBAS.

To improve the performance of CLUBAS advance text pre-processing techniques can be implemented to optimize the clustering and classification work and also modern text clustering and classification can be implemented.

Recent approaches in automatically classifying bug reports as bug and non-bug are based on text mining. Antoniol et al. [2] have investigated the automatic classification of bug reports by utilizing conventional text mining techniques, which demonstrated the feasibility. They have used title and description of bug reports.  They have collected about 1800 bug reports from Mozilla, Eclipse and Jboss. The approach consists of three pipeline phases. First, they have manually classified the bug reports as bug or non-bug. Second, the classifier is trained by the labelled data. Third, they have predicted the bug reports of test data using the trained classifier. In this paper they use Alternating Decision Tree (ADTree), Naïve Bayes (NB) and linear logistic regression classifier and evaluate their performance by using 10-fold cross validation.

Here, for logistic regression and ADTree, the precision of bug and the recall of non-bug are increased when the number of features are increased though decreased the recall for bug, for Mozilla repository. But NB exhibits limited improvement with increasing the number of features. For eclipse the performance of three classifiers is similar though the improvement is very limited with the increasing of the number of features. For JBoss the three classifiers show poor performance when the number of features are low like 20. For all repository logistic regression performs better than other classifiers.

In the following paper they classified the bug reports only as bug and non-bug. Here, further classification can be possible like as logical bug, security bug, UI related bug etc. and the performance of the classifiers are not so high. So, there are some opportunity to increase the performance.

To assigning the right bug to right developer Muhammad [3] proposed an automated approach for software bug classification. Their work can be split into three major steps pre-processing, feature selection and classification. In pre-processing step at they eliminate the stop words and punctuations. After that they use porter stemming algorithm for stemming the vocabulary. In feature selection step they use Chi-Square and Term Frequency Inverse Document Frequency. And Naive Bayes text classifier is used for bug classification.

Using the following approach they get maximum 86% accuracy. Highest accuracy is obtained when the training to testing ratio is 1:11. The following approach gives the better result than Support Vector Machine (SVM) when the training data is small. From the processing time point of view it performs better than SVM. But when the training dataset is large SVM performs better. Here the performance can be increased by using other feature selection algorithm and the synonym dictionary can be used to improve the performance.

## References

1. Nagwani, Naresh Kumar, and Shrish Verma. "CLUBAS: an algorithm and Java based tool for software bug classification using bug attributes similarities." *Journal of Software Engineering and Applications* 5.06 (2012): 436.
2. Antoniol, Giuliano, et al. "Is it a bug or an enhancement?: a text-based approach to classify change requests." *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 2008.
3. Javed, Muhammad Younus, and Hufsa Mohsin. "An automated approach for software bug classification." *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*. IEEE, 2012.