International Journal of Information Retrieval Research
Volume 12 • Issue 1

A Text Mining Framework for Analyzing Change Impact and Maintenance Effort of Software Bug Reports

Ruchika Malhotra, Delhi Technological University, India
https://orcid.org/0000-0003-3872-6213

Megha Khanna, Sri Guru Gobind Singh College of Commerce, University of Delhi, India*
https://orcid.org/0000-0003-4379-1837

ABSTRACT

Software practitioners often strive to achieve a "bug-free" software, though it is a myth. Software bug categorization (SBC) models, which assigns levels (e.g., "low," "moderate," or "high") to a software bug aid effective bug management. They assist in allocation of proper maintenance resources for bug elimination to improve software quality. This study proposes the development of SBC models that allocate levels on the basis of three software bug aspects (i.e., maintenance effort required to correct a bug, its change impact, and the combined effect of both of these). In order to develop SBC models, the authors use a text mining approach that extracts relevant features from bug descriptions and relates these features with different software bug levels. The results of the study indicate that the categorization of software bugs in accordance with maintenance effort and change impact is possible. Furthermore, the combined approach SBC models were also found to be effective.

KEYWORDS

Android Operating System, Change Impact, Empirical Validation, Machine Learning, Maintenance Effort, Software Bug Categorization, Software Quality, Text Mining

INTRODUCTION

Software bugs are inevitable. They generally get introduced during the development or maintenance phase and lead to poor quality software products with unsatisfied customers. Therefore, software maintenance activities, which remove these bugs and ensure the smooth functioning of a software are mandatory (Elmishali et al. 2018). However, it should be noted that maintenance resources are always at constraint. The software community requires to manage these resources appropriately in order to deliver timely upgrades of the software. One possible method for managing maintenance resources, while removing software bugs is Software Bug Categorization (SBC), which is explored in this study.

SBC involves cataloguing of software bugs into different levels (categories) on the basis of various bug attributes such as their criticality, priority, etc. (Menzies & Marcus 2008; Tian et al. 2015). SBC models use textual features and various other information available in the bug reports for cataloguing

DOI: 10.4018/IJIRR.295974 *Corresponding Author

them. With the aid of SBC, a software developer can take informed decisions while handling a bug and planning its correction. This study categorizes software bugs into three levels namely, "low", "moderate" and "high" on the basis of two important bug attributes, viz. Maintenance Effort (ME) and/or Change Impact (CI). The ME required to correct a software bug estimates the lines of code which are required to be added or deleted during bug correction. The CI of a bug is computed by the number of classes, which are modified while removing the corresponding bug.

It is important to categorize software bugs on the basis of ME and CI as a bug fixing regime is highly dependent on them. A software developer who is aware that a specific bug belongs to "high" category level corresponding to its ME, will allocate more effort for correction of such a bug as compared to those belonging to "low" and "moderate" levels. As a result, constraint maintenance resources can then be optimally used. Moreover, maintenance activities can complete on time within the designated budget effectively.

A bug with high CI is likely to affect the quality of the existing software product as ripple effect of its correction will be experienced by larger number of classes. This may result in deteriorating the quality of the existing system as new bugs may get introduced in other classes. If software practitioners are aware of a bug's CI, they will be careful in regression testing so that no new bugs escape into the system and are corrected as soon as possible. Moreover, if a software professional is aware that a bug belongs to "high" category with respect to its CI, he will be more careful with regression testing after bug correction and will execute a larger number of test cases as compared to those for a bug corresponding to "low" or "moderate" levels of CI. Likewise, bugs which are allocated "high" level on the basis of the combined effect of ME and CI, need both, more maintenance resources and stringent regression testing for their effective resolution. Such bugs if known can be properly allocated the required resources. Thus, planning maintenance activities based on the discussed SBC models would result in effective resolution of all categories of software bugs leading to good quality maintainable products and customer satisfiability.

Though studies in literature have developed SBC models on basis of various bug attributes such as severity, criticality, etc. (Menzies & Marcus 2008; Tian et al. 2015), the domain of bug categorization in accordance with their probable CI has not been explored. Also, the study is a first in successfully identifying the CI levels of a bug on the basis of its description. Therefore, the contributions of the study include:

- Assessment of ME levels and CI levels of software bugs on the basis of textual description present in their corresponding bug reports.
- Prediction of a software bug level on the basis of the combined effect of CI and ME to ease allocation of both maintenance as well as testing resources.
- Meticulous evaluation of results on five application packages of a popular open-source software Android, which are also statistically validated using Wilcoxon signed rank test.

In order to develop SBC models and correctly predict levels of a software bug, we extract features (words) from the bug description which are present in the bug report using a text mining approach. The SBC models are developed with six classification algorithms using Top-25, Top-50 and Top 100 words. The evaluation metrics used for assessing the performance of the developed SBC models were Area Under the Receiver Operating Characteristic Curve (AUC) and Recall. The models developed in the study can aid software managers in organizing maintenance and testing resources and improving software quality by delivering good quality products within defined timelines.

The study is organized into six sections. Section 2 describes the related literature studies. Section 3 discusses in detail the proposed SBC framework. Section 4 discusses the research methodology, while Section 5 states the results of the study followed by conclusions in the next section.

RELATED WORK

This section is divided into three parts i.e. (i) studies that have developed SBC models using bug reports, (ii) studies which have performed CI analysis using Information Retrieval (IR) techniques and (iii) studies that have assessed software ME as an ordinal variable.

Software Bug Categorization

Cataloguing software bugs and identifying their severity, priority etc. using information present in bug reports is a popular research area. Table 1 lists some of these key studies and their characteristics (features used, SBC criteria, labels, datasets and evaluation measures used). Jindal et al. (2015) have successfully attempted categorization of software bugs on the basis of ME required to correct them. However, there is a huge literature gap when it comes to categorization of software bugs into various levels on the basis of their CI. It is important to assess defects on the basis of their CI values so that the impact of bug correction may be monitored in different parts of the software and stringent regression testing may be performed, wherever required.

Table 1. Characteristics of Key Literature Studies that have performed SBC

Study	Features	SBC Criteria	Labels	Datasets	Evaluation Metric
Menzies and Marcus (2008)	Textual description of issue reports	Severity	Five Severity Levels (1,2,3,4,5)	Project and Issue Tracking System (PITS)	Recall, Precision, F-Measure
Sharma et al. (2015)	Textual summary of bug reports	Severity	Severe bugs, non-severe bugs	Eclipse	Accuracy, Precision
Chaturvedi and Singh (2012)	Textual summary of bug reports	Severity	Five Severity Levels (1,2,3,4,5)	PITS	Accuracy, F-Measure
Tian et al. (2015)	Temporal, textual, author, related-report, severity, and product factors of a bug report	Priority	Five levels (P1, P2, P3, P4, P5)	Eclipse	Recall, Precision, F-Measure
Zhang et al. (2016)	Bug report	Severity, Developer recommendation	Trivial, Minor, Major, Critical, Blocker (severity levels)	GNU Compiler Collection, OpenOffice, Eclipse, NetBeans, and Mozilla	Recall, Precision, F-Measure
Jindal et al. (2015)	Bug description	Maintenance Effort	Very low, low, medium, high	Camera application package (Android)	AUC, Recall

Change Impact Analysis using IR Techniques

Literature studies have successfully used textual information of change requests as a predictor for estimating CI sets. Poshyvanyk et al. (2009) proposed measures for analyzing conceptual coupling with the use of IR techniques for analyzing CI. Antoniol et al. (2000) mapped the maintenance request to its starting impact set using textual information present in source code and other high-level documents

of a software. A study by Zanjani et al. (2014) used interaction as well as commit histories to find the CI of an incoming change request using its textual description. Thus, we propose to predict levels of CI using textual information present in defect reports.

Software Maintenance Effort Estimation

Majority of literature studies in the domain of software maintenance effort have estimated it as a continuous variable by determining the changed number of lines of code (Malhotra and Chug 2016). Only very few studies have analyzed effort as an ordinal variable. A study by Balogh et al. (2012), predicted software development modification effort, where the target variable was analyzed as a function of weighted count of modifications and the net development time of these modifications. They were further allocated low, medium and high values by binning them into three equal sized bins. Kumar and Sureka (2017) computed the number of changed lines of code between two versions at the code tab and program organization level. However, they divided the predicted variable into three ordinal categories with the aid of fuzzy clustering algorithm. Malhotra and Lata (2020) categorized their dependent variable maintainability into two levels i.e. those requiring low maintainability effort and the others requiring high maintainability effort. Jindal et al. (2015) has ascertained the levels of ME required to correct a bug by creating ordinal variables with equal sized binning. Similar to these studies we divide our dependent variable into three categories "low", "moderate" and "high". Moreover, this study also predicts categories of software ME combined with CI by using the textual description of bug reports.

SOFTWARE BUG CATEGORIZATION FRAMEWORK

This section first discusses an overview of the proposed framework. Thereafter it discusses the steps performed in the text mining module and elaborates the evaluation of the framework.

Overview of Framework

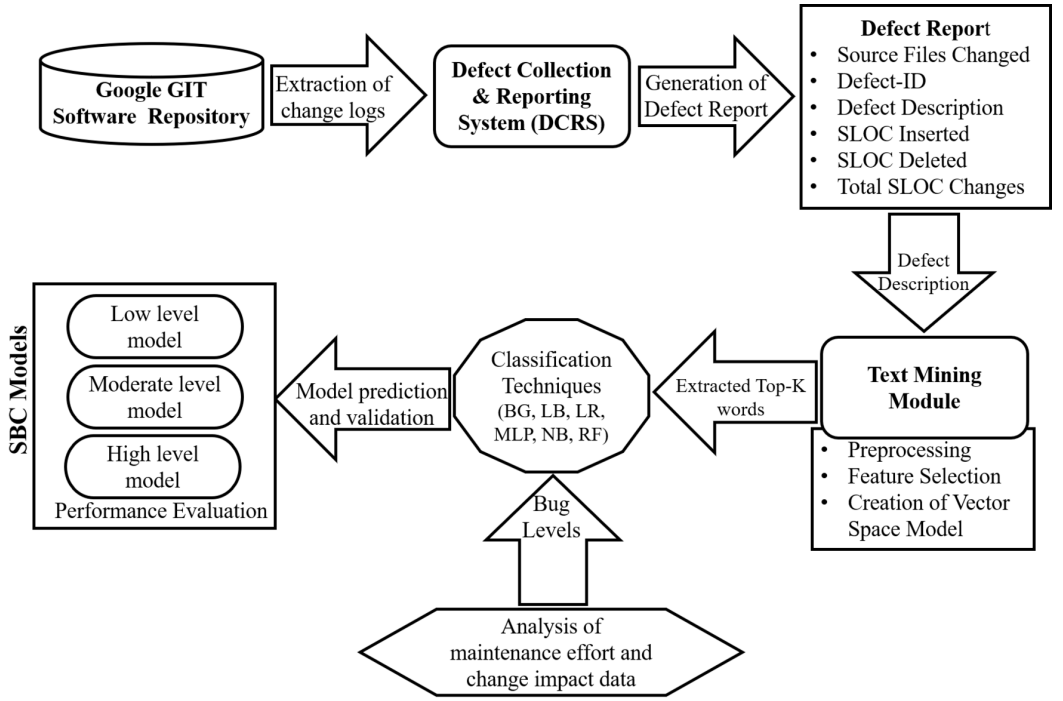
The diagrammatic representation of the framework is provided in Figure 1. We first extract the change logs available in Google's GIT software repository (https://android.googlesource.com) for five application packages of the Android Operating System (OS) so that their corresponding bug data may be analyzed. The application packages used were Bluetooth (D1), Browser (D2), Calendar (D3), Camera (D4) and MMS (D5). It must be noted that two versions of each application package were analyzed to extract the changes that have been performed in them (Table 2). Data present in the change logs between two versions will help in identifying the bugs, which have been corrected and their characteristics. For easier reference, we will be referring to application packages as D1, D2, D3, D4 and D5.

Next, we used a tool named Defect Collection and Reporting System (DCRS) to generate defect reports from the extracted change logs (Malhotra et al. 2014). Each defect report contains six fields: a) the name of the source code class (Source File Changed) b) the ID of the bug that has been removed (Defect-ID), c) the textual description of the bug (Defect description); d) the number of Source Lines of Code (SLOC) inserted while removing a bug (SLOC Inserted); the number of SLOC deleted during bug elimination (SLOC Deleted) and e) the sum of SLOC inserted or deleted for bug elimination (Total SLOC changes).

In the next step, we perform text mining steps to extract important words from textual descriptions of the bugs identified in the defect reports. The steps are explained in the next section. As an output of these steps, we obtain Top K (K= 25, 50 or 100) scoring words, which are used as independent variables while creating SBC models. We also analyze the defect reports and allocate levels to bugs on the basis of ME and CI characteristics for each software.

It should be noted that ME, the number of classes impacted and their product (combined) are continuous variables. We convert these continuous variables into ordinal variables by binning the

Figure 1. Software bug categorization framework



values into three equal sized categories according to their empirical distribution (Sentas et al. 2005). Certain previous studies have also followed this approach of creating ordinal values by binning into equal sized bins (Balogh et al. 2012; Jindal et al. 2015). The three categories formed in this study were "low", "moderate" and "high". Table 2 depicts the ranges of continuous values which are allocated to each level for a particular dataset. It also mentions the number of data points allocated to each level in a specific dataset. For instance, according to Table 2, all bugs which require [1-6.5) SLOC for correction are allocated "low" level and there are 26 such data points in D1 dataset. A data point consists of the independent variables i.e. Top K words (K=25, 50 or 100) and the allocated level (low/ moderate/ high) for a corresponding bug description.

We next develop SBC models with six classification techniques namely Random Forests (RF), Logistic Regression (LR), LogitBoost (LB), Bagging (BG), MultiLayer Perceptron (MLP) and Naïve Bayes (NB). These techniques have been simulated in the WEKA tool (Frank & Hall 2011) using its default settings. For instance, we use a learning rate of 0.3, activation function as sigmoid, one hidden layer and a momentum of 0.2 for MLP. Though, altering the parameters of the classification techniques may yield better models, exploring the entire parameter space for optimum parameter settings is difficult (Tantithamthavorn et al.) and beyond the scope of this study. Also, WEKA tool is widely accepted and is known to use reasonable parameter settings (Frank & Hall 2011). The training and validation of SBC models is done using ten-fold cross validation technique. In this case, the dataset is divided into 10 parts. Each of the 10 parts are used for testing the developed model at least once, while the remaining 90% is used for model learning. The process of training and validation is repeated 10 times.

For each dataset, three SBC (low, moderate and high) level models are developed with each of the six techniques. It should be noted that each SBC model predicts a binary outcome ascertaining whether a bug belongs to a specific level or not. For instance, all "low level" models ascertain whether

Table 2. Dataset level details

Dataset	Level	Maintenance Effort		Change Impact		Combined	
		Range	*Data Points*	*Range*	*Data Points*	*Range*	*Data Points*
Bluetooth (4.1-4.4)	Low	[1-6.5]	26	[1-1.5]	54	[1-6.5]	25
	Moderate	(6.5-39]	27	(1.5-3.5]	15	(6.5-60.5]	27
	High	>39	26	>3.5	10	>60.5	27
Browser (2.3-4.0)	Low	[1-9.5]	199	[1-1.5]	328	[1-10.5]	193
	Moderate	(9.5-38.5]	196	(1.5-2.5]	110	(10.5-66.5]	200
	High	>38.5	191	>2.5	148	>66.5	193
Calendar (4.1-4.4)	Low	[1-7.5]	56	[1-1.5]	100	[1-8.5]	55
	Moderate	(7.5-37.5]	55	(1.5-2.5]	35	(8.5-45]	55
	High	>37.5	54	>2.5	30	>45	55
Camera (2.3-4.0)	Low	[1-14.5]	118	[1-1.5]	156	[1-19.5]	116
	Moderate	(14.5-54.5]	118	(1.5-2.5]	82	(19.5-117.5]	119
	High	>54.5	117	>2.5	115	>117.5	118
MMS (2.3-4.0)	Low	[1-7.5]	54	[1-1.5]	95	[1-9.5]	56
	Moderate	(7.5-30.5]	57	(1.5-2.5]	42	(9.5-53]	56
	High	>30.5	57	>2.5	31	>53	56
D1: Bluetooth; D2: Browser; D3: Calendar; D4: Camera; D5: MMS							

bugs are associated with "low" category or "not low" category according to their corresponding textual description. Thereafter, we evaluate the developed SBC models.

Text Mining Module

The steps performed in this module can be primarily divided into preprocessing tasks, feature selection and creation of vector space model (Malhotra 2016).

Preprocessing

The primary aim of preprocessing is to decrease the size of feature space represented by the bug descriptions. This step involves tokenization, stop-word removal and stemming (Malhotra 2016; Sebastiani 2002). Tokenization includes conversion of individual characters into well-defined tokens. For this, all punctuation characters are replaced with blanks, all non-printable escape characters are dismissed and uppercase alphabets are transformed to lowercase. All the stop-words such as articles, verbs, prepositions, conjunctions, nouns, pronouns, adjectives, and adverbs are discarded in the stop-word removal step. Thereafter, stemming is performed. It involves the removal of all the words that have the same stem while retaining the stem. For instance, words like "display", "displayed", "displaying" and "displays" are removed while only the word "display" is retained. As a result of preprocessing, a reduced set of words is obtained.

Feature Selection

Though the initial set of words is reduced after preprocessing, we still need to choose an effective set of words for classification. We use infogain measure for feature selection, which ranks all the words extracted from the preprocessing step (Malhotra 2016; Sebastiani 2002). Thereafter, we extract the

top K words (K = 25, 50 or 100) based on their infogain ranks. The concept of infogain is illustrated with an example. For instance, a dataset may have 35% of "low" level bugs, 25% of "moderate" level bugs and 40% of "high" level bugs. The bug distribution D_0 of the dataset would be with bugs, b(1) = "low", b(2) = "moderate" and b(3) = "high". The corresponding frequencies are f(1) = 0.35, f(2) = 0.25 and f(3)=0.40. In order to encode bug distribution D_0 i.e. $H(D_0)$ can be defined below (Anvik et al. 2006):

$$p\left(b\right) = f\left(b\right) / \sum_{b \in D} f\left(b\right)$$

$$H\left(D\right) = -\sum_{b \in D} p\left(b\right) \log_2 p\left(b\right)$$

If S denotes the set of features, then the bits required for encoding a bug level are:

$$H\left(D|S\right) = -\sum_{s \in S} p\left(s\right) \sum_{b \in D} p\left(b|s\right) \log_2 p(b \mid s)$$

A feature S_i is ranked highest with respect to infogain if it minimizes the encoding required for the data after the feature has been used. Therefore,

$$Infogain\left(S_i\right) = H\left(D\right) - H(D \mid S_i)$$

It should be noted that we assume that the extracted features are sufficient to characterize the bug reports into different levels.

Creation of Vector Space Model

After features have been selected, we create a vector space model, which includes all the defect reports in the form of a vector. Here, each defect report is represented in the form of the top-k selected features (terms). Each of the features in a defect report (vector) is weighted using the Term Frequency- Inverse Document Frequency (TFIDF) approach. Term frequency refers to the occurrence of a feature in a specific defect report, while document frequency depicts the occurrence of a feature across all defect reports. A higher term frequency indicates higher usage of a term in a specific defect report. On the contrary, if the same term is present in many documents, it is less important due to its decreased discriminative power. A feature is weighted in accordance with high term frequency and the inverse of its document frequency. The TFIDF is defined as follows (Malhotra 2016):

$$TFIDF = T_{pq} * \log_2 \left(\frac{ND}{ND_q} \right)$$

Here, T_{pq} represents the frequency of q^{th} term in p^{th} defect report; ND is the total number of defect reports and ND_q is the total number of defect reports containing the q^{th} term. Thereafter, we normalize the feature vectors, before they are passed on to the classification techniques.

Evaluation of Framework

We evaluate the framework by analyzing the SBC models developed by it. AUC is selected as an evaluation metric because of its capability to yield stable results, even when imbalanced training data is provided (He & Garcia 2009). In order to understand AUC, we first discuss recall and specificity. The recall of a "high" level SBC model is the ratio of actual "high" level bugs to that of total bugs predicted as "high". Similarly, specificity is defined to monitor the efficacy of "not high" level predictions. "Low level" and "moderate level" models are assessed likewise. AUC plots 1-Specificity on the horizontal axis and the recall values on the vertical axis (Fawcett 2006). A higher score of AUC indicates a preferred prediction model. Furthermore, statistical evaluation is performed using Wilcoxon signed rank test. The test performs pairwise comparisons amongst SBC models. Also, the probability values are altered using Bonferroni correction The effect size of significant cases of Wilcoxon test was ascertained as suggested by Pallant 2020.

RESULTS

In order to analyze the AUC scores, we have used cut-off's suggested by Shatnawi (2012).

ME based SBC Models

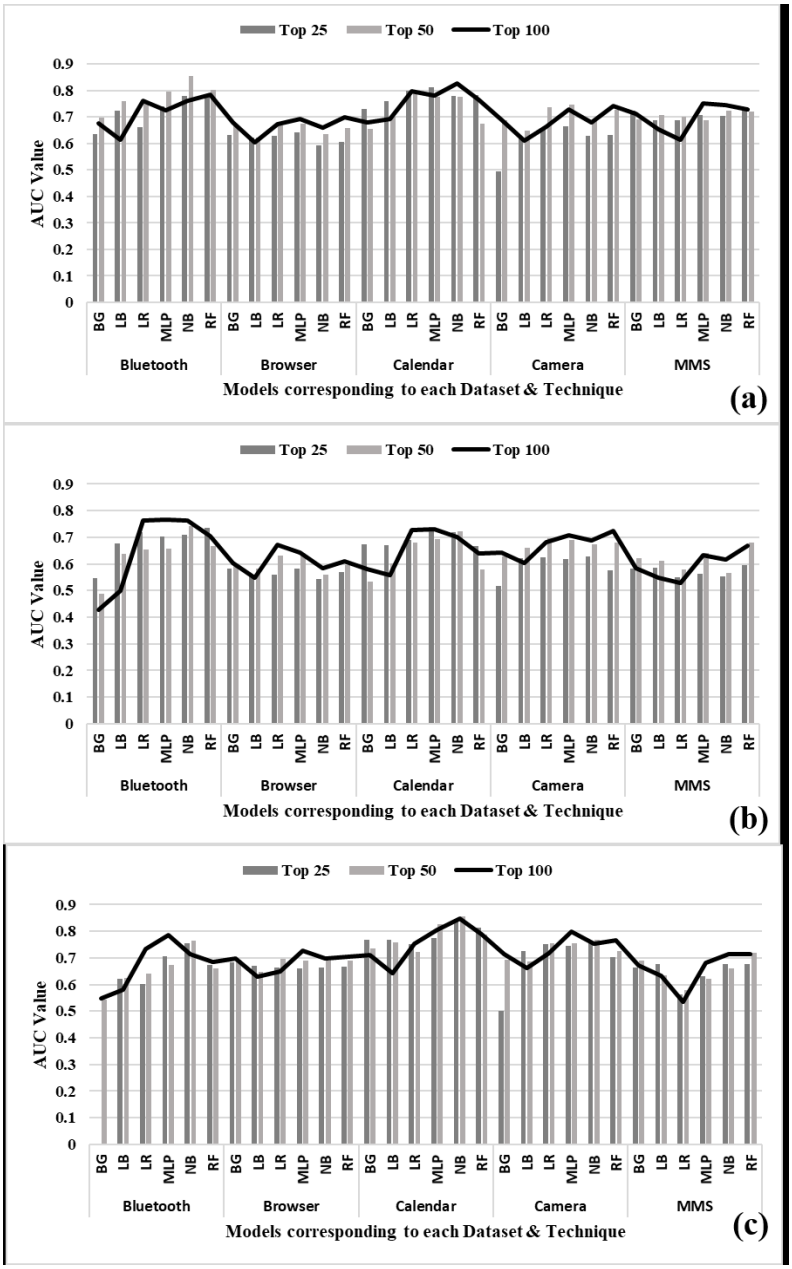
Figure 2 states the AUC scores obtained by the developed SBC models for each of the investigated classification technique (BG, LB, LR, MLP, NB, RF) on the five datasets used in the study. The majority of the AUC scores exhibited by the "low level" and the "moderate level" SBC models depicted in Figure 2(a) and Figure 2(b) respectively were in the range of 0.614-0.853 (D1), 0.543-0.698 (D2), 0.559-0.828 (D3), 0.577-0.747 (D4) and 0.549-0.750 (D5) for the corresponding datasets. According to Figure 2(c), ME based SBC models for "high level" exhibited maximum AUC scores of 0.786 (D1), 0.727 (D2), 0.856 (D3), 0.799 (D4) and 0.715 (D5) in the investigated datasets. Since the AUC scores of the majority of developed models were greater than or equal to 0.6, it signifies that ME based SBC models are accurate (Shatnawi 2012) and it is possible to catalogue bugs in accordance with the required ME.

We also assessed the average AUC scores obtained by all the six classification techniques, for a specific level and a dataset (Table 3). The best model (Top-25, Top-50 or Top-100) in a particular scenario with respect to average AUC scores is denoted in bold. It may be seen that in seven out of 15 cases, the best average scores were obtained by the Top-100 word models as they efficiently learn the keywords required to appropriately distinguish a bug report. In other cases, Top-25 and Top-50 word models were found more suitable.

On observing the average scores (AVG) of low, moderate and high level SBC models, the authors noted that "high level" models obtained better average AUC scores than the other two levels, in D3, D4 and D5 datasets. This is important as correct identification of "high level" bugs will aid in optimum resource allocation as these bugs require abundant resources for rectification, as compared to other two categories.

Figure 3 depicts the boxplots of recall values obtained by Top K (K=25, 50 or 100) SBC models for each category level, for each of the investigated classification technique. The y-axis in the figure depicts the recall values, while the x-axis represents the model developed using a specific classification technique for a particular level. Level A corresponds to "low level", Level B to "moderate level" and Level C to "high level". Thus, "BG A" represents the model developed by the BG technique for categorizing "low level" and "not low level" bugs. The average recall values on all the investigated datasets of Top-25, Top-50, and Top-100 models were 0.601, 0.628 and 0.641 respectively. The trends of recall values depicted in the boxplots indicated their increase in general, with the increase in a number of predictor variables. This is possible as a larger number of predictor variables are able to effectively encapsulate the textual description of bugs and correctly predict the appropriate level.

Figure 2.



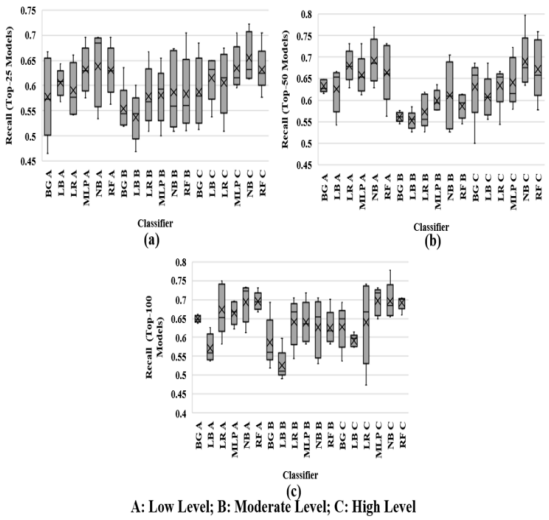
CI based SBC Models

The AUC scores of models which categorize a bug in accordance with the number of classes, which will be impacted while correcting it are depicted in Figure 4. These SBC models predict a binary outcome with respect to each level i.e. “low”, “moderate” or “high” in terms of its impact on other classes in the software system. Thus, two models are developed at each of the three levels using Top K (K=25, 50 or 100) predictors, for each of the investigated classification technique. According to the figure, the SBC models developed at “high” level (Figure 4(c)) exhibited AUC scores between

Table 3. Average AUC Scores of ME based SBC models

Dataset		Top 25	Top 50	Top 100		Top 25	Top 50	Top 100		Top 25	Top 50	Top 100
D1		0.719	**0.777**	0.720		**0.681**	0.641	0.654		0.651	0.651	**0.674**
D2		0.620	0.654	**0.668**		0.566	0.600	**0.610**		0.668	**0.684**	**0.684**
D3	Low Level	**0.777**	0.731	0.757	Moderate Level	**0.690**	0.635	0.656	High Level	**0.787**	0.781	0.758
D4		0.617	**0.707**	0.685		0.598	0.671	**0.675**		0.697	0.731	**0.735**
D5		**0.706**	0.705	0.700		0.572	**0.617**	0.597		0.648	0.650	**0.658**

Figure 3. Recall Values of (a) Top 25 (b) Top 50 and (c) Top 100 ME based SBC Models



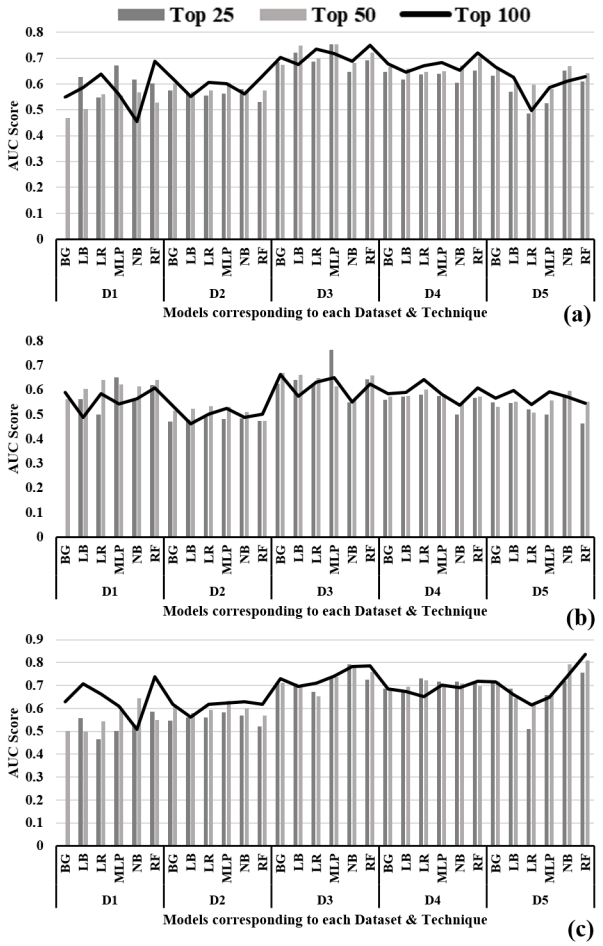
0.501-0.709 (D1), 0.522-0.624 (D2), 0.654-0.785 (D3), 0.651-0.730 (D4) and 0.510-0.835 (D5) in corresponding datasets. The SBC models at "low level" (Figure 4(a)) obtained maximum scores of 0.687, 0.622, 0.754, 0.721 and 0.668, while those at "moderate level" (Figure 4(b)) obtained maximum scores of 0.651, 0.525, 0.764, 0.642 and 0.599 respectively for D1, D2, D3, D4 and D5 datasets. It may be noted that though most of these scores (0.6-0.7) are correct and adequate, however, some SBC models exhibit AUC scores as low as 0.4 or 0.5. The reason for such weak models was unbalanced data available for training them. For instance, in D2 dataset, there were only 110 bugs with "moderate level" in the training data, while all other 476 bugs belonged to either "low" or "high" level categories (Table 2). This means that while developing binary models for "moderate" level, only 18% of the training instances were those of "moderate" category. The classification techniques were not able to effectively learn the characteristics of bugs so that they could categorize them efficiently into the "moderate" level category. This led to poor AUC scores.

We also analyzed the best model (depicted in bold) for a specific level (low, moderate or high) for all the investigated datasets using the average AUC scores exhibited by the SBC models with all the six classification techniques in Table 4. In seven cases, the Top-100 models gave superior results than Top-25, and Top-50 models. However, in three and four cases each the Top-25 and Top-50

Table 4. Average AUC Scores of CI based SBC models

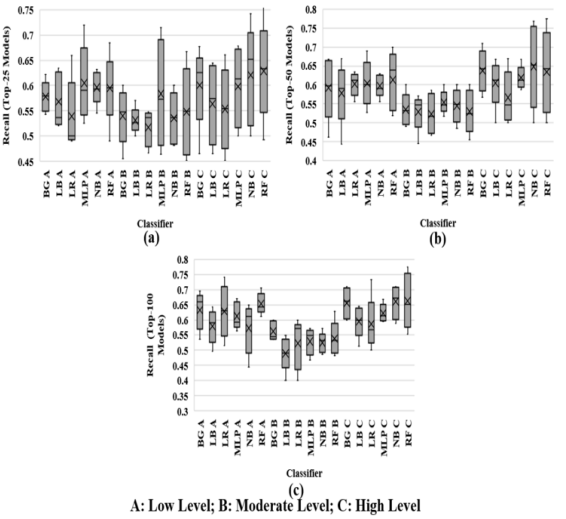
Dataset	Low Level			Moderate Level			High Level			
	Top 25	Top 50	Top 100	Top 25	Top 50	Top 100	Top 25	Top 50	Top 100
D1	**0.605**	0.531	0.579	0.583	**0.615**	0.563	0.533	0.555	**0.643**
D2	0.561	0.584	**0.595**	0.480	**0.515**	0.502	0.557	0.597	**0.612**
D3	0.698	**0.713**	0.712	**0.641**	0.636	0.616	0.724	0.725	**0.741**
D4	0.633	0.666	**0.675**	0.560	0.574	**0.591**	**0.706**	0.702	0.688
D5	0.579	**0.626**	0.603	0.527	0.549	**0.569**	0.674	**0.708**	0.703

Figure 4. AUC scores of (a) Top 25 (b) Top 50 and (c) Top 100 CI based SBC Models



models also obtained the best average AUC models indicating their suitability for identifying levels of a software bug on the basis of CI.

Figure 5. Recall Values of (a) Top 25 (b) Top 50 and (c) Top 100 CI based SBC Models



A: Low Level; B: Moderate Level; C: High Level

We also compared the average AUC of SBC models developed for identifying "low", "moderate" and "high" levels of a software bug. It was found that the "high" level bugs were categorized with higher average AUC as compared to "low" and "moderate" level categories in the majority of cases. This indicates that the bugs which have high CI values can be identified effectively. Software managers may allocate more testing resources while conducting regression testing after such bugs are corrected. This is a precautionary measure to identify that no new errors may be introduced while correcting such bugs as these bugs impact many classes during their correction.

We also analyze the recall values obtained by the developed SBC models, which categorize bugs according to CI values. The values are depicted as boxplots in Figure 5. Similar to the trend observed in ME based SBC models, the recall values of the SBC models developed to identify different levels using the investigated classification techniques improved with the increase in a number of predictor variables.

Combined Approach based SBC Models

SBC models were developed which allocated three possible levels (low, moderate or high) to a bug, according to the product of the bug's required ME and CI values. Three different models using a varied number of predictors i.e. 25, 50 and 100 were developed using each of the six classification techniques at each level. Figure 6 depicts the AUC results of the developed models. It can be seen from the figure that the AUC scores of SBC models determining "low level" bugs (Figure 6(a)) were between 0.620-0.830 (D1), 0.593-0.664 (D2), 0.668-0.814 (D3), 0.617-0.726 (D4) and 0.694-0.798 (D5) for corresponding datasets. The majority of obtained AUC scores depicted in the Figure were acceptable.

We also state the mean AUC scores depicted by the developed SBC models using the investigated classification techniques in Table 5. The mean AUC scores of "moderate level" SBC models were 0.564-0.624 (D1), 0.581-0.595 (D2), 0.656-0.680 (D3), 0.569-0.635 (D4) and 0.694-0.734 (D5). Similarly, the mean AUC scores range of "high level" SBC models were 0.564-0.606 (D1), 0.634-0.667 (D2), 0.712-0.752 (D3), 0.721-0.735 (D4) and 0.625-0.735 (D5) in the corresponding datasets. We analyzed

Figure 6. AUC scores of (a) Top 25 (b) Top 50 and (c) Top 100 Combined Approach SBC Models

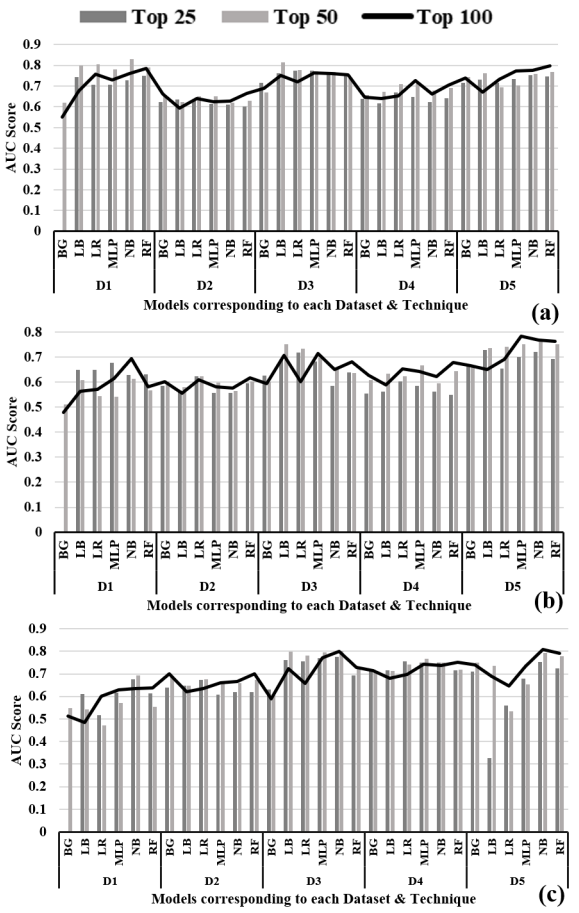
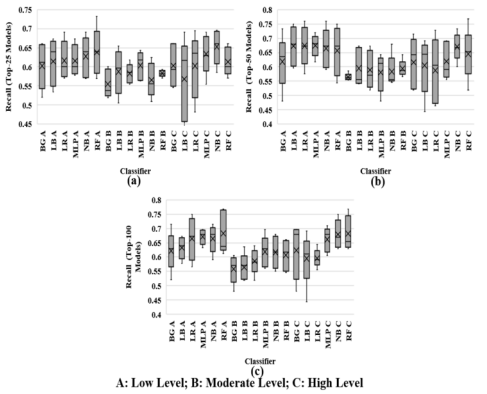


Figure 7. Recall Values of (a) Top 25 (b) Top 50 and (c) Top 100 Combined Approach SBC Models



A: Low Level; B: Moderate Level; C: High Level

the best model for a specific dataset and a specific level (low, moderate, high) using the average AUC scores. The best average AUC model for a particular scenario is depicted in bold (Table 5). According to the analysis, the Top-50 models were found best in eight out of 15 cases, followed by the Top-25 models in four cases. The Top-100 word models were designated as best in only three cases. These results indicate that it is not necessary that the increase in predictor variables would lead to an increase in AUC scores. A lot of variables could be reporting redundant information in Top-100 models. This could be a probable explanation of well performing Top-50 word models.

A comparison of average AUC scores in Table 5 indicated that in D2 and D4 datasets the "high" level category models obtained better average AUC models than the other two. It is essential to categorize "high" level bugs efficiently so that software managers can effectively plan software bug regimes, both in terms of allocation of resources during maintenance and for stringent regression testing. This would ensure development of cost-effective and good quality software product.

Table 5. Average AUC Scores of SBC models based on Combined Approach

Dataset		Top 25	Top 50	Top 100		Top 25	Top 50	Top 100		Top 25	Top 50	Top 100
D1	Low Level	0.708	**0.771**	0.710	Moderate Level	**0.624**	0.564	0.584	High Level	**0.606**	0.564	0.583
D2		0.619	**0.638**	0.635		0.581	**0.595**	0.590		0.634	**0.667**	0.664
D3		**0.758**	0.756	0.740		0.656	**0.680**	0.658		0.731	**0.752**	0.712
D4		0.640	**0.687**	0.671		0.569	0.629	**0.635**		**0.735**	0.733	0.721
D5		0.733	0.738	**0.749**		0.694	**0.734**	0.720		0.625	0.707	**0.735**

The recall values of the developed SBC models were also evaluated. They are depicted as boxplots in Figure 7. The maximum recall values obtained by Top K (K=25, 50 or 100) models were 0.732, 0.768 and 0.768 respectively. Moreover, it was noted that there was an increase in recall values with the increase in the number of predictor variables. The average recall value of all the Top-100 word models was computed as 0.629.

DISCUSSION

The effectiveness of SBC models was compared using Wilcoxon signed rank test. The pairwise performance of the SBC models which categorized bugs in accordance with the combined approach was compared with the other two SBC models (ME based SBC models and CI based SBC models) at all the three levels ("low", "medium" and "high"). We conduct the Wilcoxon test on average AUC scores obtained by all the investigated classification techniques (BG, LB, LR, MLP, NB and RF) for Top K (K=25, 50 or 100) words models on all the five datasets of the study. The hypothesis is evaluated at a significance level of $\alpha = 0.05$.

According to the results of the Wilcoxon test based on average AUC scores we found that the performance of the combined approach SBC models was better than CI based SBC models at all the three levels, more so significantly in two levels (low & moderate). The effect size of the Wilcoxon test for significant cases was found to be large (0.5-0.6), indicating the superiority of the combined approach as compared to CI based SBC models. The underlying reason for this observation was the imbalanced nature of the training data for CI based SBC models. A careful analysis of Table 2 indicates that majority of software bugs in the training data of all the five datasets belonged to "low" category (D1: 68%, D2: 56%, D3: 61%, D4: 44%, D5:56%), when they were allocated levels based

on CI. This is because most of the software bugs were resolved by making modifications in just one class. Thus, the classification techniques were not able to learn "moderate" and "high" level instances appropriately indicating slightly poor performance of these developed SBC models as compared to other (based on combined approach or ME) developed SBC models.

Wilcoxon test results on average AUC scores indicated that the performance of combined approach SBC models was comparable to ME based SBC models as the difference was not found significant at any level. However, it should be noted that combined models are more useful as they take into account both the ME and CI values. Thus, they aid in the planning of both developer manpower as they are capable of forecasting levels of a bug on the basis of ME, as well as tester's effort as they are able to forecast bugs which influence change in a large number of classes. A tester may focus their effort on executing test cases of those classes which are affected by a change to ensure that the software is functioning smoothly.

Apart from analyzing the Top-25, Top-50 and Top-100 word models, we also assessed Top-10 word models. However, Top-10 models showed poor performance as compared to other investigated models. Thus, we did not include their values in the study. The key reason for this outcome would be the inadequacy of Top-10 words to encapsulate the requisite information required for categorizing a software bug. Another interesting observation indicated that both Top-50 and Top-100 models exhibited effective AUC scores. As previously discussed, the probable reason for such an outcome would be redundancy in the information represented by the top-100 predictors.

We also performed a Wilcoxon signed rank test at a cut-off of 0.05 to compare all the developed SBC models "level" wise. We compared the average AUC scores of "high" level models obtained using Top-25, Top-50 and Top-100 words for all the three discussed bug categorizing approaches with "low" and "moderate" category models in order to ascertain which "level" developed SBC models were most accurate. The Wilcoxon test results indicated that the performance of "high" level models was better than "low" level models and significantly better than "moderate" level models. This indicates that the "high" category bugs are identified with higher accuracy and thus, such bugs should be first determined and allocated appropriate maintenance and testing resources. Such a practice would aid software managers is maintaining good quality software in a cost-effective manner. Thereafter, the remaining resources should be distributed amongst "low" and "moderate" category bugs.

The results of the study are unique as only work by Jindal et al. (2015) developed ME based SBC models. Studies in literature have not explored the development of CI based or combined approach based SBC models. Moreover, Jindal et al. only validated one dataset and one classification technique as compared to five datasets and six techniques used in this study. Jindal et al. reported a recall of 0.81 and an AUC score of 0.89 for "high level" models developed by Top-100 words. This study reports a maximum recall of 0.70 and a maximum AUC score of 0.86 for the same models. Though, there is a slight decrease in the maximum values, the results of this study are more generalizable as higher number of datasets and classification techniques have been evaluated and discussed. Furthermore, this study recommends the use of combined approach models as they are more useful than SBC based models based only on ME.

CONCLUSION

This study proposes an approach to predict the levels of required ME and the probable CI of a bug based on textual description present in the corresponding bug report. Binary models were developed to identify three levels i.e. "low", "moderate" and "high" for a specific bug. The models indicated the level either based on (i) only ME (ii) only CI or (iii) product of (i) and (ii). The study assessed three possible number of predictors i.e. 25, 50 and 100 which were top words extracted from bug report using text mining. Empirical validation was performed using five application packages of Android software. The SBC models were developed using six classification techniques and the results were validated using AUC scores and Recall values. The primary results are stated as follows:

- It is important to prioritize maintenance and testing resources by correctly identifying "high" level models. Thus, we outline the average AUC scores obtained by these models on all the five datasets using all the investigated techniques. The average AUC scores of ME based models with Top-100 words as predictors ranged from 0.658-0.758. Similarly, the AUC scores for SBC models that identified high level bugs based on CI using Top-100 words were in the range 0.612-0.741. The average AUC scores of SBC models in majority of the datasets that identified "high" level bugs in accordance with combined influence of CI and ME were in the range 0.664-0.735. The average recall values of all Top-100 models using all the classification techniques were 0.657, 0.629 and 0.639 for SBC models based on ME, CI and the combined approach respectively. These trends indicate the acceptable capability of the developed SBC models. Similar trends were observed by the "moderate" category and "low" category SBC models.
- The accuracy (in terms of AUC scores) of the combined approach SBC models were found to be statistically superior to CI based SBC models. SBC models based on combined approach did not exhibit any statistical difference when compared with SBC models that allocated levels based on ME.
- The accuracy of "high" category models was found statistically superior to "moderate" and "low" category models when compared using average AUC scores. It was also assessed that both Top-50 and Top-100 models exhibit better AUC scores than Top-25 word models. Though, higher number of predictors improves the AUC score, one should be careful that the selected predictors should not represent redundant information. Thus, software managers have to make a strategic decision while choosing K=50 or 100 for developing effective SBC models.

The developed SBC models, especially the ones that predict bugs belonging to "high level" in accordance with both ME as well as CI (i.e. combined approach) are beneficial in outlining efficient resource usage. This is because high level bugs, once correctly identified, should be allocated higher percentage of the available resources both during testing as well as maintenance phase. This would enable effective bug resolution and cost-effective development of a good quality maintainable software. Software practitioners can also efficiently plan bug fixing regimes as a bug which is categorized as "high" with respect to the required maintenance effort but is exhibiting "low" change impact values will focus on changes in few classes. Such bugs may require more developer effort but low testing effort.

REFERENCES

Antoniol, G., Canfora, G., Casazza, G., & De Lucia, A. (2000, March). Identifying the starting impact set of a maintenance request: A case study. In *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering* (pp. 227-230). IEEE. doi:10.1109/CSMR.2000.827331

Anvik, J., Hiew, L., & Murphy, G. C. (2006, May). Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering* (pp. 361-370). doi:10.1145/1134285.1134336

Balogh, G., Végh, Á. Z., & Beszédes, Á. (2012). Prediction of software development modification effort enhanced by a genetic algorithm. *SSBSE Fast Abstrackt Track*, 1-6.

Chaturvedi, K. K., & Singh, V. B. (2012, September). Determining bug severity using machine learning techniques. In *2012 CSI sixth international conference on software engineering (CONSEG)* (pp. 1-6). IEEE. doi:10.1109/CONSEG.2012.6349519

Elmishali, A., Stern, R., & Kalech, M. (2018). An artificial intelligence paradigm for troubleshooting software bugs. *Engineering Applications of Artificial Intelligence*, *69*, 147–156. doi:10.1016/j.engappai.2017.12.011

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. doi:10.1016/j.patrec.2005.10.010

Frank, E., & Hall, M. A. (2011). *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284. doi:10.1109/TKDE.2008.239

Jindal, R., Malhotra, R., & Jain, A. (2015, August). Mining defect reports for predicting software maintenance effort. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 270-276). IEEE. doi:10.1109/ICACCI.2015.7275620

Kumar, L., & Sureka, A. (2017, February). Using structured text source code metrics and artificial neural networks to predict change proneness at code tab and program organization level. In *Proceedings of the 10th Innovations in Software Engineering Conference* (pp. 172-180). doi:10.1145/3021460.3021481

Malhotra, R. (2016). *Empirical research in software engineering: concepts, analysis, and applications*. CRC Press. doi:10.1201/b19292

Malhotra, R., & Chug, A. (2016). Software maintainability: Systematic literature review and current trends. *International Journal of Software Engineering and Knowledge Engineering*, *26*(08), 1221–1253. doi:10.1142/S0218194016500431

Malhotra, R., & Lata, K. (2020). An empirical study on predictability of software maintainability using imbalanced data. *Software Quality Journal*, *28*(4), 1581–1614. doi:10.1007/s11219-020-09525-y

Malhotra, R., Pritam, N., Nagpal, K., & Upmanyu, P. (2014, September). Defect collection and reporting system for git based open source software. In *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)* (pp. 1-7). IEEE. doi:10.1109/ICDMIC.2014.6954234

Menzies, T., & Marcus, A. (2008, September). Automated severity assessment of software defect reports. In *2008 IEEE International Conference on Software Maintenance* (pp. 346-355). IEEE. doi:10.1109/ICSM.2008.4658083

Pallant, J. (2020). *SPSS survival manual: A step by step guide to data analysis using IBM SPSS*. Routledge.

Poshyvanyk, D., Marcus, A., Ferenc, R., & Gyimóthy, T. (2009). Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, *14*(1), 5–32. doi:10.1007/s10664-008-9088-2

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47. doi:10.1145/505282.505283

Sentas, P., Angelis, L., Stamelos, I., & Bleris, G. (2005). Software productivity and effort prediction with ordinal regression. *Information and Software Technology*, *47*(1), 17–29. doi:10.1016/j.infsof.2004.05.001

Sharma, G., Sharma, S., & Gujral, S. (2015). A novel way of assessing software bug severity using dictionary of critical terms. *Procedia Computer Science*, *70*, 632–639. doi:10.1016/j.procs.2015.10.059

Shatnawi, R. (2012, March). Improving software fault-prediction for imbalanced data. In 2012 international conference on innovations in information technology (IIT) (pp. 54-59). IEEE. doi:10.1109/INNOVATIONS.2012.6207774

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016, May). Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 321-332). doi:10.1145/2884781.2884857

Tian, Y., Lo, D., Xia, X., & Sun, C. (2015). Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, *20*(5), 1354–1383. doi:10.1007/s10664-014-9331-y

Zanjani, M. B., Swartzendruber, G., & Kagdi, H. (2014, May). Impact analysis of change requests on source code based on interaction and commit histories. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (pp. 162-171). doi:10.1145/2597073.2597096

Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software*, *117*, 166–184. doi:10.1016/j.jss.2016.02.034

Ruchika Malhotra is a Professor and HOD in Department of Software Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She is Associate Dean in Industrial Research and Development, Delhi Technological University. She has been awarded prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from the Department of Computer and Information Science, Indiana University-Purdue University Indianapolis (2014-15), Indianapolis, Indiana, USA. She received her master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She is the recipient of Commendable Research Award by Delhi Technological University for her research in the year 2017, 2018, 2019 and 2020. She is the author of book titled ``Empirical Research in Software Engineering`` published by CRC press and co-author of a book on ``Object Oriented Software Engineering`` published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics. Her h-index is 33 as reported by Google Scholar. She has published more than 200 research papers in international journals and conferences.

Megha Khanna, also known as Megha Ummat, is currently working in Sri Guru Gobind Singh College of Commerce, University of Delhi. She completed her doctoral degree from Delhi Technological University in 2019 and her master's degree in software engineering in 2010 from the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. She received her graduation degree in Computer Science (Hons.) in 2007 from Acharya Narendra Dev College, University of Delhi. She has been the recipient of Commendable Research Award by Delhi Technological University consecutively in the years 2017, 2018 and 2019. She was also awarded the "Research Incentive" twice for her research in the year 2018 and 2019 by the Governing body of Sri Guru Gobind Singh College of Commerce. Her research interests are in software quality improvement, applications of machine learning techniques in change prediction, and the definition and validation of software metrics. Her h-index is 9 as reported by Google Scholar. She has various publications in international conferences and journals. She can be contacted by email at meghakhanna86@gmail.com.