Predicting Blocking Bugs with Machine Learning Techniques: A Systematic Review

Selasie Aformaley Brown[1], Benjamin Asubam Weyori[2]
Adebayo Felix Adekoya[3], Patrick Kwaku Kudjo[4], Solomon Mensah[5]
Department of Computer Science, University of Energy and Natural Resources, Sunyani-Ghana[1, 2, 3]
Department of Information Technology, University of Professional Studies Accra-Ghana[1, 4]
Department of Computer Science, University of Ghana, Legon-Ghana[5]

Abstract—**The application of machine learning (ML) techniques to predict blocking bugs have emerged for the early detection of Blocking Bugs (BBs) in software components to mitigate the adverse effect of BBs on software release and project cost. This study presents a systematic literature review of the trends in the application of ML techniques in BB prediction, existing research gaps, and possible research directions to serve as a reference for future research and an application insight for software engineers. We constructed search phrases from relevant terms and used them to extract peer-reviewed studies from the databases of five famous academic publishers, namely Scopus, SpringerLink, IEEE Xplore, ACM digital library, and ScienceDirect. We included primary studies published between January 2012 and February 2022 that applied ML techniques to building Blocking Bug Prediction models (BBPMs). Our result reveals a paucity of literature on BBPMs. Also, previous researchers employed ML techniques such as Decision Trees, Random Forest, Bayes Network, XGBoost, and DNN in building existing BB prediction models. However, the publicly available datasets for building BBPMs are significantly imbalanced. Despite the poor performance of the *Accuracy* metric where imbalanced datasets are concerned, some primary studies still utilized the *Accuracy* metric to assess the performance of their proposed BBPM. Further research is required to validate existing and new BBPM on datasets of commercial software projects. Also, future researchers should mitigate the effect of class imbalance on the proposed BB prediction model before training a BBPM.**

Keywords—Blocking bugs; systematic review; software maintenance; bug report; reliability; machine learning

I. Introduction

Software bugs are inevitable occurrences in the development and maintenance of software products. Hence, software engineers rely on bug reports generated by bug tracking tools such as Bugzilla, BugHead, and Trac to manage and resolve errors in software components, an activity aimed at improving and maintaining the quality of software projects. In Bugzilla, for instance, errors encountered by a software tester or user are logged in the bug tracking system and assigned the state NEW. The state of the bug report then changes to ASSIGNED when the bug is allocated to a suitable developer to resolve. Once the bug is fixed, a developer other than the one to whom the bug was assigned then ascertains the fixing and closes the bug report. At this point, the state of the bug report changes to CLOSED. However, the status of a bug report may remain at the ASSIGNED state for a long time because of another bug preventing the bug from being resolved. This type of bug is referred to as a Blocking bug [1]. In this context, blocking bugs are defined as bugs that prevent other bugs from being fixed. Thus, the time for fixing such a bug depends greatly on how long it will take to detect and resolve the blocking bug. Previous work by Valdivia-Garcia and Shihab [2] confirms that fixing a BB is almost three times the amount of time needed to fix a non-BB. Consequently, the debugging process may be impacted negatively, affecting software release and increasing the cost of software maintenance. Furthermore, Bohm et al. [3] found that locating and fixing a bug in a software product after the deployment stage is about 100 times more expensive than addressing it during the early phase of the software development life cycle. Hence, the early detection of Blocking Bugs (BBs) in software projects is critical to software maintenance. While there are criteria for detecting BBs in bug reports, the method is manual and heavily reliant on the bug reporters' and the developer assigned's competence in providing suitable labels [1]. However, the skills of the software user or bug reporter to accurately label a bug as BB is intrinsically in doubt, hence the heavy dependence on the developer (i.e., to whom the bug was assigned) to label such bugs.

Additionally, the unstructured nature of the text in some bug reports makes the manual BB classification process by developers laborious and error-prone. Meanwhile, large software projects are likely to have enormous bug reports; for instance, Valdivia-Garcia et al. [1] collected 609,800 bugs from eight software projects, out of which 77,448 were blocking bugs and 532,352 non-blocking bugs. The researchers further discovered that manually identifying blocking bugs takes 3–18 days. Therefore, the over-reliance on software developers prolongs the process of identifying BBs in bug reports and is also time-consuming as the number of bug reports increases. These fundamental challenges present the opportunity to apply ML techniques to predicting BBs (i.e., classifying and detecting BBs). Although some peer-reviewed articles have been published on using ML techniques to predict BBs [1], [2], [4]-[7], there is a dearth of literature on the subject. For instance, a thorough search in databases of well-known publishers such as Scopus, SpringerLink, IEEE Xplore, ACM digital library, and ScienceDirect produced only six papers. Unfortunately, none of these six papers was an SLR. Yet, SLR is crucial in a specific domain of studies for discovering research questions and rationalizing future research [8]. Even though a recent SLR[9] on the broader topic of bug severity acknowledged BBs as a severe bug, there has been no SLR published on the specific area of applying ML techniques to predicting BBs since the field was first found introduced in 2014 [1]. Thus, to promote further research and increase the quality of literature in this domain, a systematic review that comprehensively discusses the existing BBPMs, research gaps, and possible research directions to serve as a point of reference for research and practice is critically important. Additionally, this research will contribute to a better understanding of the trends in characterizing and predicting blocking bugs. In other words, the goal of this study is to find out the recent trends and directions in this field and to identify opportunities for future research by researchers and practitioners within the software engineering domain and also to appreciate how the research space has evolved over time with regards to BBs. Finally, the approach presented in this study can serve as a guide for researchers and practitioners (e.g. Ph.D. Student, Master students) when seeking to predict instances of bugs that are BBs based on various data miners.

Based on the aforementioned needs and motivations, in this work, we systematically present a detailed analysis of the trends in the application of ML techniques in BB prediction. Thus, the study aims at answering the following five research questions (RQs):

RQ1: What are the publication trends in BB prediction research?

RQ2: Which datasets are used to train the proposed prediction model?

RQ3: What kind of ML learning techniques is adopted in building the proposed BB prediction model?

RQ4: Which evaluation criterion is used to measure the performance of the BB prediction model?

RQ5: Which ML classifiers are used as baselines to benchmark the proposed model?

The remainder of study is organized as follows: Section 2 describes the proposed research method, including the overall process and the goal and research questions addressed in this study. Section 3 discusses the findings of the meta-analysis of the study. Section 4 presents the results of the systematic. Section 5 summarizes and concludes the study and provides future research directions.

II. Research Method

This SLR follows the software evidence-based engineering (SEBE) guidelines proposed by kitchenham and Charters [10]. The SEBE guidelines have increasingly gained popularity and acceptance in the software engineering research space [11]-[16]. The SEBE provides an all-inclusive outline of how software engineering researchers can conduct SLR using evidence-based research and practice models. Therefore, we segmented the SLR into four major phases with subdivisions to conform with the prescriptions by kitchenham and Charters [10]: plan search, search procedure, search, and report. Fig. 1 depicts the SLR process of this study.

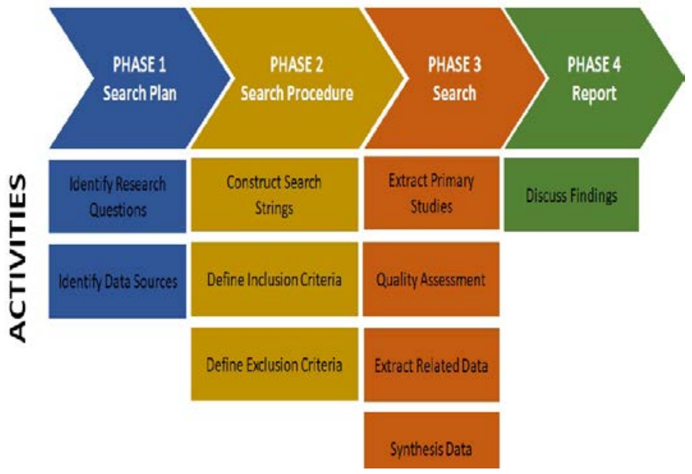


Fig. 1. Process Flow Diagram.

A. Phase 1: Search Plan

This phase of the SLR process presents the research questions addressed in this study and the databases where primary studies were collected.

1) Data sources: This section captures the academic databases and repositories where the search was conducted. Five databases of famous academic publishers, namely Scopus, SpringerLink, IEEE Xplore, ACM digital library, and Science Direct, were the data sources for the collection of primary studies for this paper. Also, a few portions of publications were retrieved from Google Scholar to achieve thorough coverage of article collection. Table I displays sampled data sources and the number of results returned by search queries in those academic databases on 18th February 2022.

B. Phase 2: Search Procedure

This phase of the SLR describes how search strings were constructed. It also captures the inclusion and exclusion criteria used in selecting primary studies.

TABLE I. Data Sources and Search Results

Data Source	**URL**	**Result**
Scopus	https://scopus.com	61
SpringerLink	https://link.springer.com	246
IEEE Xplore	https://ieeexplore. ieee.org	20
ACM Digital Library	https://dl.acm.org	152
ScienceDirect	https://ieeexplore.ieee.org	31
Google Scholar	https://scholar.google.com	169
	Total Result	**679**

1) *Search string*: Keywords were extracted from research questions and related papers. Then synonyms and alternate words were identified for creating search strings by using Boolean *OR* for alternative words and Boolean *AND* for linking search keywords. The keywords and their alternatives are shown in Table II. The resulting search phrases were tweaked to conform with the format required by each online database. Table III shows the search string for each database. Additionally, the Table IV shows the search strings per the data sources that were used in this study.

2) *Inclusion criteria*: Peer-reviewed articles about blocking bugs published in journals, conferences, technical reports, or book chapters from 2012 to 2022 were targeted for review. Whereas 2014 marked the first application of machine learning to predict blocking bugs [1], 2012 was chosen as the start date to widen the scope of our search. Also, the focus was on publications in English that used ML to detect and classify blocking bugs or predict a phenomenon related to blocking bugs. Moreso, studies that constructed prediction models based on binary data classification of blocking bugs were selected. The authors ensured that all selected publications reported their data source, performance evaluation, baseline techniques, and the challenges and limitations of their studies.

3) *Exclusion criteria*: Mendeley [17], a reference management software, was used to delete duplicate papers which were 274 in number. Next, the papers that were not peer-reviewed and for which the complete text was not available in English were excluded. An article that was not about blocking bugs in computer software or was not written regarding predicting a phenomenon of blocking bugs with machine learning was not considered. Articles were mainly excluded based on titles and abstracts, full-text reading, and later quality evaluation. The data sources and the corresponding number of publications after the exclusion criteria are captured in Table III.

TABLE II. KEYWORDS AND ALTERNATIVE WORDS/PHRASES

Keyword	**Alternative word/phrase**
Blocking Bug	('blocker bugs' OR 'severe bugs' OR 'Bug Severity')
Prediction	('Identifying' OR 'Classifying' OR 'Detection' OR 'Characterizing')
Machine Learning	('machine technique' OR 'method' OR 'model' OR 'algorithm')

TABLE III. NUMBER OF PUBLICATIONS AFTER EXCLUSION CRITERIA

Data Source	**Number of shortlisted Publications**
Scopus	2
SpringerLink	0
IEEE Xplore	3
ACM Digital Library	1
ScienceDirect	0
Google Scholar	0
Total	6

TABLE IV. SEARCH STRINGS PER DATA SOURCE

Data Source	**Search String**
Scopus	("Blocking Bug" OR "Severe Bugs" OR "Bug Severity" OR "Blocker Bug") AND ("Identifying" OR "Prediction" OR "Classifying" OR "Detection" OR "Characterizing") AND ("machine learning" OR "method" OR "model" OR "algorithm")
SpringerLink	("Blocking Bug" OR "Severe Bugs" OR "Bug Severity" OR "Blocker Bug") AND ("Identifying" OR "Prediction" OR "Classifying" OR "Detection" OR "Characterizing") AND ("machine learning" OR "method" OR "model" OR "algorithm")
IEEE Xplore	("Blocking Bug" OR "Severe Bugs" OR "Bug Severity" OR "Blocker Bug") AND ("Identifying" OR "Prediction" OR "Classifying" OR "Detection" OR "Characterizing") AND ("machine learning" OR "method" OR "model" OR "algorithm")
ACM Digital Library	[[All: "blocking bug"] OR [All: "severe bugs"] OR [All: "bug severity"] OR [All: "blocker bug"]] AND [[All: "identifying"] OR [All: "prediction"] OR [All: "classifying"] OR [All: "detection"] OR [All: "characterizing"]] AND [[All: "machine learning"] OR [All: "method"] OR [All: "model"] OR [All: "algorithm"]]
ScienceDirect	("Blocking Bug" OR "Severe Bugs" OR "Blocker Bug") AND ("Prediction" OR "Classifying" OR "Detection" OR "Characterizing") AND ("machine learning" OR "algorithm")
Google Scholar	("Blocking Bug" OR "Blocker Bug") AND ("Identifying" OR "Prediction" OR "Classifying" OR "Detection" OR "Characterizing") AND ("machine learning" OR "method" OR "model" OR "algorithm")

C. *Phase 3: Search*

This phase of the SLR explains the approach adopted for sampling relevant primary studies.

1) *Study selection*: To sample relevant primary studies that meet the needs of this study, the tollgate method proposed by Afzal et al. [16] was adopted. Fig. 2 depicts the tollgate approach used. This approach is made up of five steps which were traced by the authors as follows:

Step 1: Data was collected from selected online data sources via the use of search strings generated in Table III.

Step 2: Duplicate studies were excluded using Mendeley.

Step 3: Inclusion/ exclusion criteria were applied by perusing the titles and abstracts.

Step 4: Inclusion/ exclusion criteria were applied by reading the introductions and conclusions.

Step 5: Inclusion/ exclusion criteria were applied by reading the full text of sampled studies.

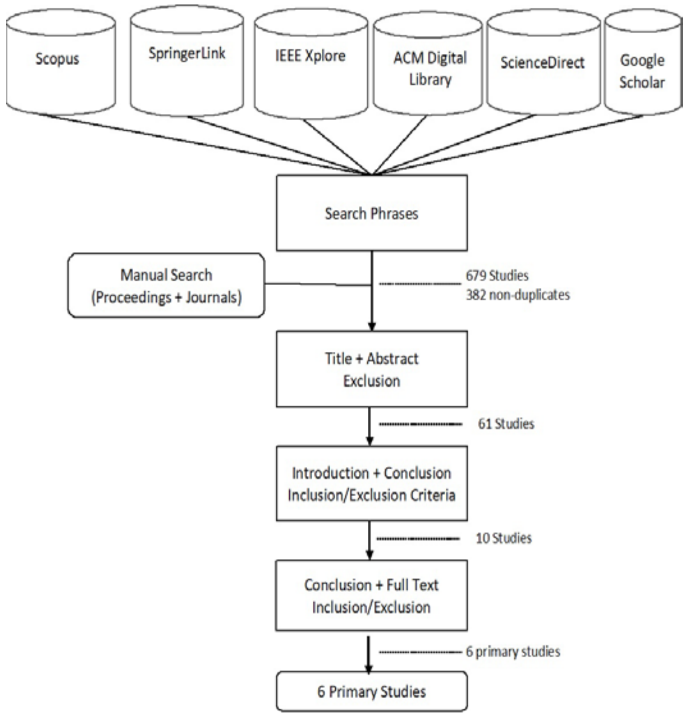


Fig. 2. The Tollgate Approach.

The search strings established in Table III were used to collect 679 from the selected data sources for this study (i.e. Table I) in the first step. At the end of the tollgate approach, six primary studies [1], [4]-[7] were selected. Table VII shows the shortlisted primary studies.

2) Quality assessment of selected studies: The purpose of the study quality assessment was to ensure that the sampled primary studies could adequately answer the research questions. It also aids in interpreting findings for data analysis and synthesis [18]. Hence, each quality criterion was designated by the prefix ‘QC’ and a number. The QC and data extraction activities were performed at the same time. Table V shows a quality assurance checklist for selected primary studies. A mathematical approach was used to assign quality scores, as in Kitchenham [10]. Thus, attributes of the quality criteria outlined in **Table V** were extracted for each primary study and scored on how well they met the requirements. For each attribute there are three possible values: Yes (Y) = 1 point, Partial (P) = 0.5 point, No (N) = 0 point. A study is allocated a score of 1 if the article clearly answers the QC question and a score of 0.5 if it partially answers the QC. Studies that do not answer the QC questions receive a score of 0. The six QC scores were added together to get the total quality score. As a result, the overall quality score of each selected study ranged from 0 (extremely poor) to 6 (very good). This approach to quality score has been widely used by SLR researchers in the software engineering domain [14], [19], [20], and related domains [21]. Each selected article in this study received a score greater than 70%. This percentage score shows that the primary studies can sufficiently answer the research questions.

TABLE V. QUALITY CHECKLIST FOR SELECTING PRIMARY STUDIES

Serial Number	QC Checklist
QC1	Does the selected study give details of the machine learning techniques applied in the study to answer RQ1?
QC2	Does the selected study give details of the dataset and the data source used in the study to answer RQ2?
QC3	Does the selected study benchmark its results with the performance of other prediction techniques to answer RQ3?
QC4	Does the selected study provide information about the performance metrics used to evaluate results to answer RQ4?
QC5	Does the selected study report the ML classifiers used as baselines to benchmark the proposed model to answer RQ5?

3) Data extraction: A structured extraction form created with Microsoft Excel was used to extract the information needed for data synthesis. Table VI indicates the items extracted from each primary study.

TABLE VI. DATA EXTRACTION FORM ITEMS

Data Item	Description
Reference	Title, Author,Type (i.e Journal/conference/workshop)
Technique	ML technique was applied in building the proposed model in the study.
Pre-processing	Preprocessing methods for machine learning technique
Dataset	Source of datasets used in training ML models
Evaluation	Metrics used for model evaluation
Results	The outcome of the model performance evaluation
Baseline	Baseline techniques with which proposed models were compared
Future works	Future works proposed by the study

TABLE VII. SHORTLISTED PRIMARY STUDIES

Ref.	Year	Library	Journal/Conference
Valdivia-Garcia and Shihab [23]	2014	ACM	Conference: MSR’14, May 31 – June 1, 2014, Hyderabad, India
Xia et al	2015	Elsevier	Journal: Information and Software Technology
Valdivia-Garcia et al[2]	2018	Elsevier	Journal: Journal of Systems and Software
Din et al.	2020	IEEE Xplore	Conference: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)
Cheng et al. [24]	2020	IEEE Xplore	Conference: IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)
Brown et al. [25]	2021	IEEE Xplore	Conference: 2021 International Conference on Cyber Security and Internet of Things (ICSIoT)

4) Data synthesis: At this stage, the relevant extracted data were synthesized using the thematic approach [22] to answer the research questions outlined in Table V.

D. Phase 4: Report

The final phase of the SLR summarizes and examines the review results. Then, in distinct sub-sections, the full summary of the findings of this review is discussed and interpreted in relation to the research questions.

III. Findings

RQ1: What are the publication trends in BB prediction research?

Our search following the SEBE SLR methodology identified six primary studies that applied ML to predicting BB. The publication period of these studies spans from January 2014 to February 2022. Table VII presents the selected publications, while Fig. 3 depicts the publications over the years or distribution of primary studies. Observing Table VII, conference publications or proceedings seem to dominate the publications over the period with four articles, while journal articles account for two publications. This distribution further suggests the paucity of publications on the use of ML for predicting BBs over the years.

RQ2: Which datasets are used to train the proposed prediction model?

The performance of a machine learning technique is heavily reliant on the quality of the dataset used in training the prediction model. To train BBPM, all the six selected primary studies in this work utilized datasets extracted from publicly available bug reports about specific software projects. These bug reports were retrieved from Bugzilla, IssueTracker, and monorail issue tracking systems. The ones obtained from Bugzilla are Eclipse, NetBeans, Gentoo, Fedora, Mozilla, and NetBeans. While bug reports of Chromium and OpenOffice were retrieved from Montrail and IssueTracker. Table IX shows the web locations where these bug reports about the various projects were extracted. Also, studies used bug reports from authentic projects with actual proportions of blocking bugs (BBs) and non-blocking bugs (Non-BBs). Table VIII and Fig. 4 show blocking bug and non-blocking bug distribution within the extracted datasets and the distribution of projects from which datasets were extracted per the studies, respectively.

From Table VIII, it is observed that the chosen primary studies made use of at least two sets of datasets from the open-source application domain. For instance, Ding et al. [5] validated their method on two open-source projects, namely, Mozilla Firefox and Netbeans, which contained 132,584 bugs. 18900 were Blocking Bugs and 113,684 Non-Blocking Bugs. Also, Cheng et al. [7] gathered a total of 214873 bugs from Eclipse, FreeDesktop, NetBeans, and OpenOffice, of which 16,402 were Blocking Bugs. Both Valdivia-Garcia and Shihab [1] and Xia et al. [4] selected a total of 402,962 bugs from six (6) open-source projects (i.e. Chromium, Eclipse, FreeDesk Mozilla, NetBeans, and OpenOffice). 18,422 of the total bugs were blocking bugs. Similarly, Valdivia-Garcia et al. [2] and Brown et al. [6] mined a total of 609,800 bugs which had 77,448 blocking bugs from eight projects (i.e., Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, OpenOffice, Gentoo, and Fedora) in their studies. The open-source projects used by the selected primary studies in this paper, as well as their corresponding bug tracking systems, are as follows:

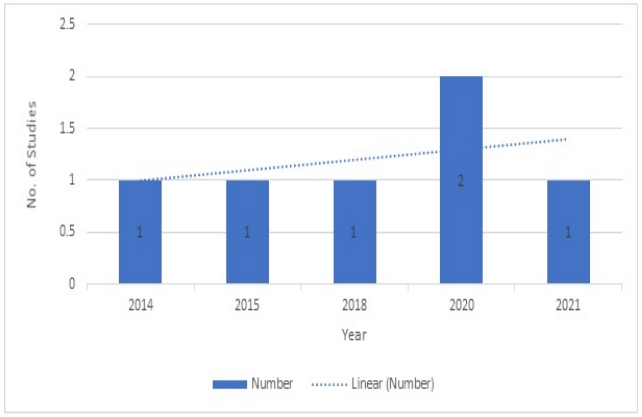


Fig. 3. Distribution of Primary Studies.

TABLE VIII. Distribution of Dataset Utilized by Primary Studies

Studies	No. of Projects	Projects	No. of Bbs	No. of Non-Bbs
Valdivia-Garcia and Shihab [1]	6	Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, and OpenOffice	18,422	384,540
Xia et al. [12]	6	Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, and OpenOffice	18,422	384,540
Valdivia-Garcia et al. [11]	8	Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, OpenOffice, Gentoo, and Fedora	77,448	532,352
Cheng et al. [14]	4	Eclipse, FreeDesktop, NetBeans, and OpenOffice	34,892	229,729
Ding et al. [13]	2	Mozilla Firefox and Netbeans	16,402	198,471
Brown et al. [15]	8	Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans, OpenOffice, Gentoo, and Fedora	18,900	113,684

TABLE IX. SOURCES WHERE DATASETS OF OPEN-SOURCE PROJECTS WERE EXTRACTED

Project	Source
Chromium	https://bugs.chromium.org/p/chromium/issues/list
Eclipse	https://bugs.eclipse.org/bugs/
NetBeans	https://netbeans.org/bugzilla/
OpenOffice	https://bz.apache.org/ooo/
Gentoo	https://bugs.gentoo.org/buglist
Fedora	https://bugzilla.redhat.com/ss
Mozilla	https://bugzilla.mozilla.org/
free desktop	https://bugzilla.freedesktop.org/

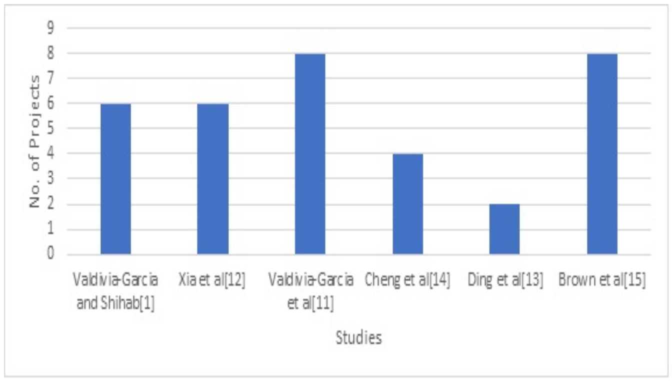


Fig. 4. Distribution of Projects Studied by the Selected Studies.

- Chromium is a popular open-source web browser developed by Google and used mainly as the codebase for Google Chrome. C++ and C programming languages dominate it; however, it comprises other programming languages such as JavaScript and Python, amongst others. Chromium has its bug tracking mechanism in Google code, called Monorail, which has a feature called "Blocking." The "blocking" feature can identify if a bug is a blocking bug or not.

- Eclipse is a well-known integrated development environment (IDE) developed mainly with Java yet supports many programming languages, including Python, Ruby, and C/C++. In addition, eclipse uses Bugzilla for reporting and tracking bugs. This issue tracking system has a feature called "Blocks." This is used for identifying a bug as a blocking bug. GNU/Linux or FreeBSD. Gentoo also uses Bugzilla in reporting and tracking bugs; hence the "Blocks" field identifies a blocking bug.

- Mozilla is an open-source project that hosts and develops products such as Firefox, Thunderbird, Bugzilla, Gecko layout engine, and others. The programming languages used in its development are C, JavaScript, and C++. In addition, Mozilla tracks its bugs in Bugzilla software and uses the "Blocks" field to show if a bug is a blocking bug.

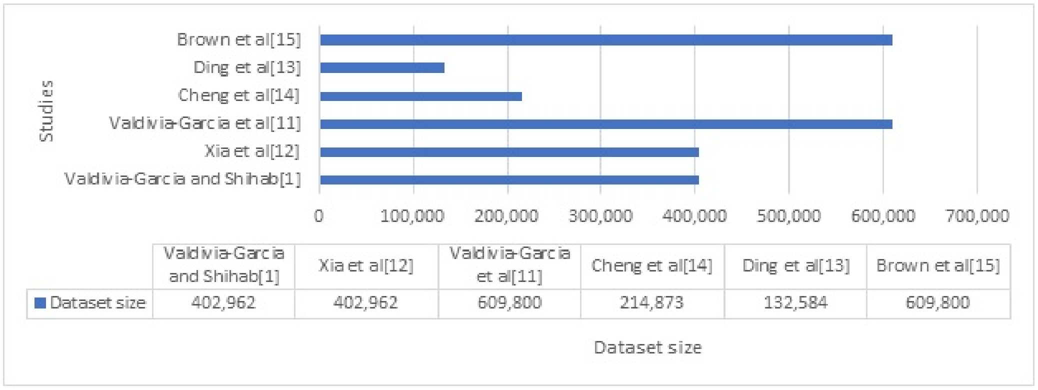
- NetBeans is an open-source IDE for developing applications in the java programming language for Windows, Mac, Linux, and Solaris. However, it supports PHP, C, and C++, amongst others. NetBeans was developed with the Java programming language. Bugzilla is an issue tracking system used by Netbeans; hence the "Blocks" field indicates whether a reported bug is a blocking bug or not.

- OpenOffice is an office suite created by Sun Microsystems and is now maintained by Apache. It is maintained with its programming language called OpenOffice.org Basic. IssueTracker, a derivative of Bugzilla, was used by OpenOffice when these primary studies were undertaken. Just like Bugzilla, it has a "Blocks" feature which indicates whether a bug is a blocking bug or not. At the time of this study, Bugzilla had succeeded IssueTracker, which was also known as IssueZilla.

Valdivia-Garcia et al. [2] and Brown et al. [6] used the most extensive dataset with the most bugs (i.e. 609,800 bugs), which they extracted from Chromium, Eclipse, Free Desktop, Mozilla, NetBeans, OpenOffice, Gentoo, and Fedora. The open-source project that most of the studies included in their dataset is Netbeans, while Gentoo and Fedora were the least utilized by the studies, as captured in Fig. 6. Although the six primary studies used datasets from popular and well-supported open-source projects with a substantial number of bug reports, the distribution of dataset sizes in Fig. 5, coupled with the unequal distribution of Blocking bugs and non-blocking bugs in Table VIII, suggests the challenge of class imbalance. Thus, in most studies, BBs account for less than 12% of the total available dataset [2]. However, extensive research exists about the challenges an imbalanced dataset, also referred to as the class imbalance phenomenon [23], poses to the performance of classifiers that use them in training. Also, it is worth noting that all the studies considered in this work used datasets related to open-source projects; hence their findings may not apply to closed and commercial software projects.

RQ3: What kind of ML learning techniques is adopted in building the proposed BB prediction model?

The primary studies considered in this SLR proposed new methods for identifying a bug as blocking bugs or non-blocking bugs based on an existing classification technique. Fig. 7 shows the distribution of the classification techniques employed in the various studies.



	Valdivia-Garcia and Shihab[1]	Xia et al[12]	Valdivia-Garcia et al[11]	Cheng et al[14]	Ding et al[13]	Brown et al[15]
■ Dataset size	402,962	402,962	609,800	214,873	132,584	609,800

Fig. 5. Distribution of Datasets by Selected Studies.

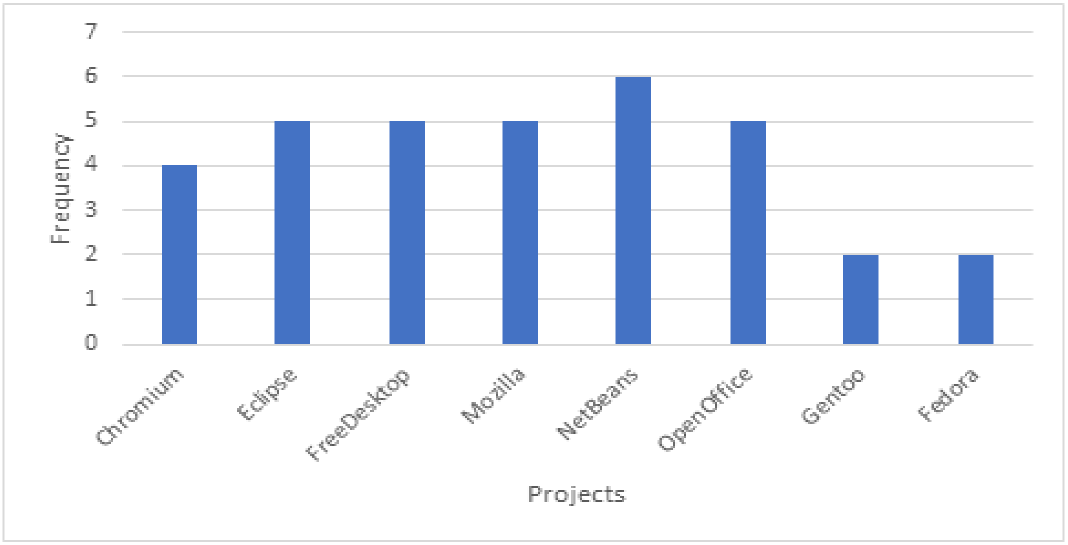


Fig. 6. Distribution of Projects and their Frequency of use in Primary Studies.

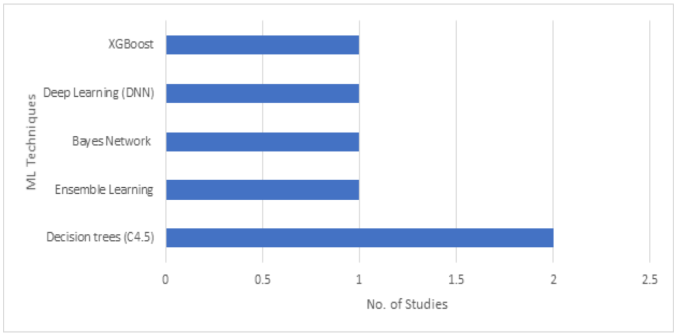


Fig. 7. Distribution of Classification Techniques.

TABLE X. DISTRIBUTION OF ML TECHNIQUES UTILIZED BY PRIMARY STUDIES

ML Techniques	**Studies**
Decision trees (C4.5)	Valdivia-Garcia and Shihab[1], Valdivia-Garcia et al [11]
Ensemble learning	Xia et al. [12]
Bayes Network	Ding et al. [13]
Deep Learning (DNN)	Brown et al. [15]
XGBoost	Cheng et al. [14]

Valdivia-Garcia and Shihab [1] and Valdivia-Garcia et al. [2] used a re-sampling technique to pre-process the training data to resolve the data imbalance issue; non-blocking bugs outnumber blocking bugs. Even though random forest performed better in terms of F1 measure than their proposed model, which was based on Decision trees (C4.5), they recommended their model as the most appropriate for practitioners.

Xia et al. [4] built a classifier called ELBlocker based on the random forest technique. They separated the training data into many disjoint sets and developed different classifiers, which they then merged to identify an appropriate threshold for classifying bugs as blocking or non-blocking. Also, Cheng et al. [7] presented a new classification framework called XGBlocker, consisting of two stages. XGBlocker captures more features from bug reports in the first stage to construct an improved dataset. In the second stage, XGBlocker employs the XGBoost technique to build an efficient model for performing the prediction task. Ding et al. [5] proposed a Bayes Network-based classifier for forecasting the breakability of the blocking bug pairs.

The classifier is identical to the Bayes Network classifier; as the threshold lowers from 0.5 to 0, the classifier becomes stricter and removes more boundary instances to improve precision. In Brown et al. [6], researchers proposed a DeepLabb classifier based on deep neural networks for predicting blocking bugs. Three deep neural networks were developed and trained independently, each with a different number of hidden layers. The first DNN had two hidden layers, the second DNN had three hidden layers, and the third DNN had four hidden layers. Bayesian optimization was used to estimate the best learning rate for each model. It is worth mentioning that apart from Brown et al. [6], who employed a deep learning approach in this domain, at the time of this work, the rest used Decision Trees, Random Forest, Bayes Network, and XGBoost for building Blocking bug prediction models in the selected primary studies. Table X shows the studies and the classification techniques adopted in building their classifiers. While the most frequently used ML technique in building BB classifiers is based on the Decision Trees, Random Forest was reported by two studies [1][11] to have performed better than Zero-R, Naive Bayes, and KNN.

RQ4: Which evaluation criterion is used to measure the performance of the BB prediction model?

The primary studies assessed the prediction abilities of their proposed BB prediction model using various combinations of evaluation metrics. Fig. 8 depicts the distribution of research based on performance metrics. In Valdivia-Garcia and Shihab [2], Precision, Recall, F1-Score, and Accuracy were used to measure the effectiveness of their proposed classifier and reported 9-29% precision, 47-76% recall, and 15-42% F1-Score.

The F1 score and cost-effectiveness were used to assess the efficiency of ELBlocker in Xia et al. [12], which attained an F1-Score up to 0.482 and EffectivenessRatio@20% scores of 0.831.In Valdivia-Garcia et al. [11], the researchers utilized Precision, Recall, and F1-Score to evaluate the performance of their proposed model. The proposed model achieved 13%–45% precision, 47%–66% Recall, and 21%–54% F1-Score. Ding et al. [13] used ROC Area, Accuracy, F1-Score, Recall, and Precision to assess the efficiency of the classifier they recommended. In Mozilla Firefox the proposed BayesNet model achieved 0.629, 0.729, 0.676, 0.831, and 76.54 % for Precision, Recall, F-measure, Roc Area, and Accuracy, respectively. However, it recorded Precision, Recall, F-measure, Roc Area, and Accuracy of 0.488, 0.583, 0.531, 0.764, and 73.45%, respectively, in the case of the NetBeans dataset. To compare the performance of XGBlocker to other classifiers, Cheng et al. [14] employed AUC, Cost-Effectiveness, and F1-Score. XGBlocker reported an F1-score of 0.808, ER@20% of 0.944, and AUC of 0.975.

Brown et al [15] used MCC, F1, and AUC to compare DeepLaBB with other classifiers. DeepLaBB recorded an MCC of 0.8504%, F1 Score of 0.4292%, and AUC of 2.9459%.

The following is a summary of the various performance metrics used in the primary studies considered in this work:

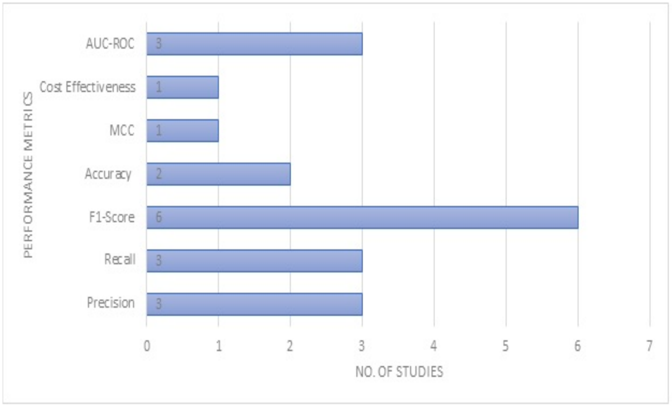


Fig. 8. Performance Metrics and the Corresponding Number of Primary Studies.

Accuracy refers to the proportion of correctly categorized instances to the total number of instances. It can be calculated using the formula below with the aid of True positives (TP), False negative (FN), False positives (FP), and True negative (TN) extracted from the confusion matrix.

Accuracy= (TP+TN)/ (TP+FP+ TN+ FN)

MCC refers to Matthew's Correlation Coefficient. To measure the quality of binary categorization, MCC examines all true and false positives and negatives [29]. It can be computed as:

MCC= (TP*TN-FP*FN)/√ ((TP+FP) (TP+FN) (TN+FP) (TN+FN))

Recall is the ratio of accurately categorized positive cases to the total number of positive instances. Recall can be calculated as:

Recall= TP/ (TP+FN)

Precision is a measure of the proportion of correctly categorized positive instances among all positive samples. It can be computed as follows:

Precision= TP/ (TP+FP)

F1-Score if the harmonic mean of precision and recall. F1's best value is 1, and its worst value is 0. It can be represented mathematically as:

F1-Score= (2*Precision*Recall)/ (Precision Recall)

The AUC-ROC refers to AUC (Area under Curve)-ROC (Receiver Operating Characteristic). It is a trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) and represents the classifier's ability to predict classes correctly.

It can be generated by charting TPR (True Positive Rate), i.e., Sensitivity or recall vs. FPR (False Positive Rate), i.e., 1-Specificity, at different threshold values.

]

Cost-Effectiveness [24] is a cost-sensitive indicator of prediction performance. It assesses a method's prediction performance under a cost limit. Even though research findings [25] discourage the use of Accuracy for evaluating classifiers trained with an imbalanced dataset, Valdivia-Garcia and Shihab [1] and Ding et al. [24] employed Accuracy in their works; the F1 score is the most common evaluation metric utilized among the studies in this SLR, followed by Precision and Recall. The least common measure employed is cost-effectiveness.

RQ5: Which ML classifiers are used as baselines to benchmark the proposed model?

The studies considered in this work compared the performance of proposed BB prediction models to baseline techniques to assess their efficacy. Fig 8 shows the baselines to which the proposed techniques were compared. After reviewing the primary studies in this work, it can be concluded that the proposed BB prediction models outperformed individual baseline classifiers in the vast majority of cases at the time of this study. The C4.5 Decision tree algorithm-based BB prediction model proposed in Valdivia-Garcia and Shihab [1] and Valdivia-Garcia et al. [2] performed better than Naive Bayes, kNN, and Zero-R baselines.

The C4.5 based model was chosen over the random forest because it is an explainable model that practitioners can easily understand. To forecast the possibility of a blocking bug, Xia et al. [4] used an ensemble of various classifiers. The ELBlocker showed a significant improvement compared with Valdivia-Garcia and Shihab's methods [1], SMOTE, one-sided selection (OSS), and bagging. In Cheng et al. [7], the proposed XGBlocker was compared with Gradient Boosting Decision Tree (GBDT), AdaBoost, ELBlocker, XGB_14, CART, Logistic Regression, and Valdivia-Garcia and Shihab's approaches. The proposed method displayed superior performance in all instances. Ding et al. [5] offered a method for describing and forecasting the breakability of the blocked bug pairs, which performed better compared with the Zero-R Classifier's performance, Naïve Bayes, BayesNet, KNN, and Random Forest. Brown et al. [6] introduced DeepLaBB for predicting blocking bugs in open-source projects. DeepLaBB showed improved performance compared with the performance of Random Forest, KNN, CART, and ANN on the same datasets. The performance of proposed BB predicting models was compared with that of base classifiers such as RF, KNN, NB, Zero classifier, and Valdivia-Garcia and Shihab's approaches in the majority of the research articles. Generally, the choice of baseline varied from one study to another. However, as shown in Fig. 9, RF was the most utilized baseline classifier in most studies, closely followed by KNN. Even though most of the proposed BB prediction models in the various studies performed better than the baseline classifiers, some BB prediction models have not improved performance compared with traditional classifiers. For instance, in Valdivia-Garcia and Shihab [1], when it comes to chromium and Eclipse data sets, the proposed model had recall values of 49% and 47%, slightly below the 50% recall value of the baseline. In the same paper, random forest performed better than the proposed model in precision across all project datasets. Also, Zero-R outperformed all the classifiers in terms of Accuracy. Similarly, in Valdivia-Garcia et al. [1], the Zero-R model had the highest Accuracy across all project datasets except for Fedora. Also, in Brown et al. [6], a baseline classifier, Random Forest, performed better than the proposed DeepLaBB in the FreeDesktop dataset in terms of MCC and F1-Score.

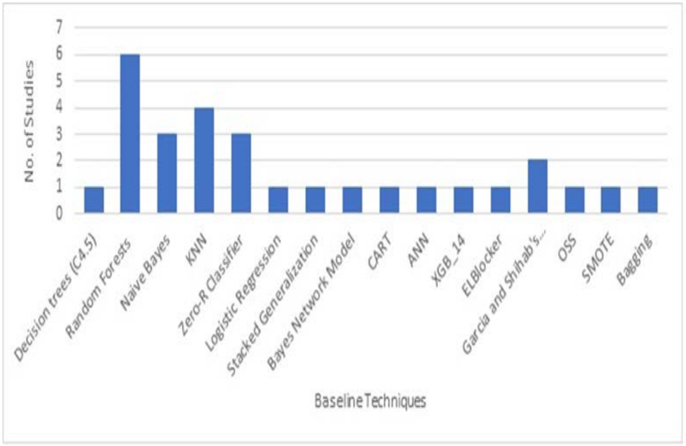


Fig. 9. Distribution of Baseline Classifiers and Primary Studies.

IV. Summary

This SLR traced the research advances in applying ML techniques to predict Blocking Bugs. After a rigorous analysis of the most pertinent research papers published between January 2012 to February 2022 in the databases of five famous academic publishers, namely Scopus, SpringerLink, IEEE Xplore, ACM digital library, and Science Direct, six (6) BB primary papers/studies were identified and reviewed. The findings regarding the research trend, variety of proposed ML techniques, baseline classifiers, evaluation metrics, and sources of datasets for predicting BBs during this study are captured in Fig. 3, Table X, Fig. 9, Fig. 8, and Table IX, respectively. The existing studies confirm that proposed ML techniques (i.e. BBPMs) significantly improve the detection of BBs in software bug reports and that they generally outperform the traditional classifiers. Also, this study concludes that there is a paucity of literature on the application of ML to BB prediction. Further research is required to validate existing and new prediction models on bug reports of commercial or closed software projects. In addition, new researchers should explore the effect of parameter tuning and the efficiency of ML approaches such as deep learning and ensemble learning in improving the classification of BBs. Furthermore, before training a classifier, researchers should take steps to mitigate the effect of class imbalance on the proposed BB prediction model.

References

[1] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in Proceedings of the 11th working conference on mining software repositories, 2014, pp. 72-81.

[2] H. Valdivia-Garcia, E. Shihab, and M. Nagappan, "Characterizing and predicting blocking bugs in open source projects," Journal of Systems and Software, vol. 143, pp. 44-58, 2018.

[3] B. W. Boehm, "Software engineering economics," in Software pioneers, ed: Springer, 2002, pp. 641-686.

[4] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," Information and Software Technology, vol. 61, pp. 93-106, 2015.

[5] H. Ding, W. Ma, L. Chen, Y. Zhou, and B. Xu, "Predicting the breakability of blocking bug pairs," in 2018 IEEE 42nd Annual Computer

Software and Applications Conference (COMPSAC), 2018, pp. 219-228.

[6] S. A. Brown, B. A. Weyori, A. F. Adekoya, P. K. Kudjo, S. Mensah, and S. Abedu, "DeepLaBB: A Deep Learning Framework for Blocking Bugs," in 2021 International Conference on Cyber Security and Internet of Things (ICSIoT), 2021, pp. 22-25.

[7] X. Cheng, N. Liu, L. Guo, Z. Xu, and T. Zhang, "Blocking Bug Prediction Based on XGBoost with Enhanced Features," in 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), 2020, pp. 902-911.

[8] P. V. Torres-Carrión, C. S. González-González, S. Aciar, and G. Rodríguez-Morales, "Methodology for systematic literature review applied to engineering and education," in 2018 IEEE Global engineering education conference (EDUCON), 2018, pp. 1364-1373.

[9] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," Information and software technology, vol. 115, pp. 58-78, 2019.

[10] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, et al., "Systematic literature reviews in software engineering–a tertiary study," Information and software technology, vol. 52, pp. 792-805, 2010.

[11] M. Niazi, A. M. Saeed, M. Alshayeb, S. Mahmood, and S. Zafar, "A maturity model for secure requirements engineering," Computers & Security, vol. 95, p. 101852, 2020.

[12] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, "Variability in software systems—a systematic literature review," IEEE Transactions on Software Engineering, vol. 40, pp. 282-306, 2013.

[13] M. Ilyas, S. U. Khan, and N. Rashid, "Empirical validation of software integration practices in global software development," SN Computer Science, vol. 1, pp. 1-23, 2020.

[14] R. Hoda, N. Salleh, J. Grundy, and H. M. Tee, "Systematic literature reviews in agile software development: A tertiary study," Information and software technology, vol. 85, pp. 60-70, 2017.

[15] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Aligning software engineering education with industrial needs: A meta-analysis," Journal of Systems and Software, vol. 156, pp. 65-83, 2019.

[16] J. dos Santos, L. E. G. Martins, V. A. de Santiago Júnior, L. V. Povoa, and L. B. R. dos Santos, "Software requirements testing approaches: a systematic literature review," Requirements Engineering, vol. 25, pp. 317-337, 2020.

[17] "Search | Mendeley." https://www.mendeley.com/search/ (accessed Apr. 04.

[18] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[19] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. r. Mahrin, "A systematic literature review of software requirements prioritization research," Information and software technology, vol. 56, pp. 568-585, 2014.

[20] M. Goulão, V. Amaral, and M. Mernik, "Quality in model-driven engineering: a tertiary study," Software Quality Journal, vol. 24, pp. 601-633, 2016.

[21] A. Qazi and N. A. Fayaz Hussain, "Rahim, Glenn Hardaker, Daniyal Alghazzawi, Khaled Shaban, Khalid Haruna 2019," Towards Sustainable Energy: A Systematic Review of Renewable Energy Sources, Technologies, and Public Opinions," IEEE Access, vol. 7, pp. 63837-63851.

[22] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in 2011 international symposium on empirical software engineering and measurement, 2011, pp. 275-284.

[23] H. He, "Member, IEEE, and Edwardo A. Garcia," Learning from Imbalanced Data,"" IEEE Transactions on knowledge and data engineering, vol. 21, pp. 1041-4347, 2009.

[24] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 432-441.

[25] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, "Assessing predictors of software defects," in Proc. Workshop Predictive Software Models, 2004.