A Survey on Machine Learning-based Automated Software Bug Report Classification

Ömer Köksal
ASELSAN Research Center
Ankara, Turkey
koksal@aselsan.com.tr

Ceyhun Emre Öztürk
ASELSAN Research Center
Department of Electrical and Electronics Engineering, Bilkent University
Ankara, Turkey
ceyhunozturk@aselsan.com.tr

2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) | 978-1-6654-7013-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISMSIT56059.2022.9932822

***Abstract*—In software development processes, classifying software bugs is a vital step since it helps grasp the nature, implications, and causes of software failures. Further, categorization enables reacting to software bugs appropriately and faster. However, manual classification of software bugs is inefficient and costly, especially in large-scale software projects, since one must deal with extensive bug reports from multiple sources. Hence, many studies have addressed this problem by automated software bug classification with the help of machine learning techniques. Researchers used various machine learning-based algorithms and techniques to obtain better classification performance. Furthermore, many researchers used open source bug repositories to compare their results with previous studies. In this paper, we aimed to report the main studies in machine learning-based automated software bug report classification by highlighting the recent improvements and indicating the key steps in this process. So, this survey can benefit the researchers and practitioners working in automated software bug report classification and other related domains.**

Keywords—software engineering, software bug report classification, machine learning, deep learning, natural language processing

I. INTRODUCTION

Software failures are almost unavoidable in contemporary software systems by their complicacy and enormous size. Therefore, software verification and validation processes are getting more important in software development, taking more time and effort. In software verification and validation processes, understanding and resolving the root cause of the faults are as important as detecting the failures to ensure the software projects' functional and quality requirements. Detected faults and the steps which led to the failure are described in the software bug reports that contain textual explanations. Since software bug reports contain crucial information about failures, the completeness and quality of these reports are decisive in software verification and validation steps. Also, for developers to fix the bug, the main source of understanding the failure and its cause heavily depends on the quality of the software bug reports.

In this paper, we focus on bug report classification. The bugs can be categorized into different types using the information stated in bug reports. Several bug schemas were proposed in the literature to categorize bugs into various types. One of the oldest bug schemas was proposed by Bezier [1]. Later, Chillarege [2] offered another schema named Orthogonal Defect Categorization to classify software bugs. Finally, another schema is proposed by Seamon [3]. These schemas were used in several types of research to classify the bug types in various software development projects. With the rise of artificial intelligence (AI), various types of research were performed using machine learning (ML) based classifiers, data/text mining, natural language processing (NLP) techniques, and bug classification schemas. In the next sections, we will provide detailed information on these works.

II. BACKGROUND

This paper focuses on the ML-based automatic classification of software bug reports. Hence, first, we briefly mention the ML and NLP techniques and the algorithms used in the bug report classification process.

A. Machine Learning

ML is a crucial part of AI that uses historical data to enable software to predict accurately. Hence, ML algorithms are categorized by their learning types: Methods in which the algorithm is trained with the labeled data are called supervised learning. Whereas in unsupervised ML, no labeled data is used. The unsupervised ML algorithm splits data using the features extracted from the data. In reinforcement learning, the agents are trained to make a sequence of decisions. The training is based on rewarding desired behaviors and punishing others.

Deep learning (DL) is an important subclass of ML based on artificial neural networks. With their improved learning and feature extraction capabilities, DL algorithms provide superior results in almost all AI domains [4], [5].

ML is widely used in automated software bug report classification. Firstly, traditional ML algorithms such as naïve Bayes (NB), K-nearest neighborhood (KNN), logistic regression (LR), support vector machines (SVM), decision tree (DT), random forest (RF), and their variants were used in software bug report classification. Later, several DL-based algorithms (multi-layer Perceptron (MLP), convolutional neural networks (CNN), long short-term memory (LSTM), and gated recurrent unit (GRU), were used in this domain.

B. Natural Language Processing

NLP uses several ML, DL, and AI-based algorithms [6]. Further, NLP uses many information retrieval and data/text mining techniques. NLP is one of the core technologies used in software bug report classification and related tasks. Apart from ML and DL algorithms, several word embedding libraries (such as Word2Vec [7] and FastText [8]) and transformer algorithm-based pretrained language models (such as BERT [9] and ELECTRA [10]) are widely used in the software bug classification domain.

978-1-6654-7013-1/22/$31.00 ©2022 IEEE

III. Software Bug Classification Process

We extract the main steps of the software bug classification process. Although these steps may vary depending on the application, the essential sub-processes are revealed in Figure 1.

We have modeled this process in Business Process Model and Notation (BPMN) format [11], as shown in Figure 1. Although we present the software bug classification steps in a simplified format, they are good to follow practically. Hence in this work, we follow the steps given in this figure.

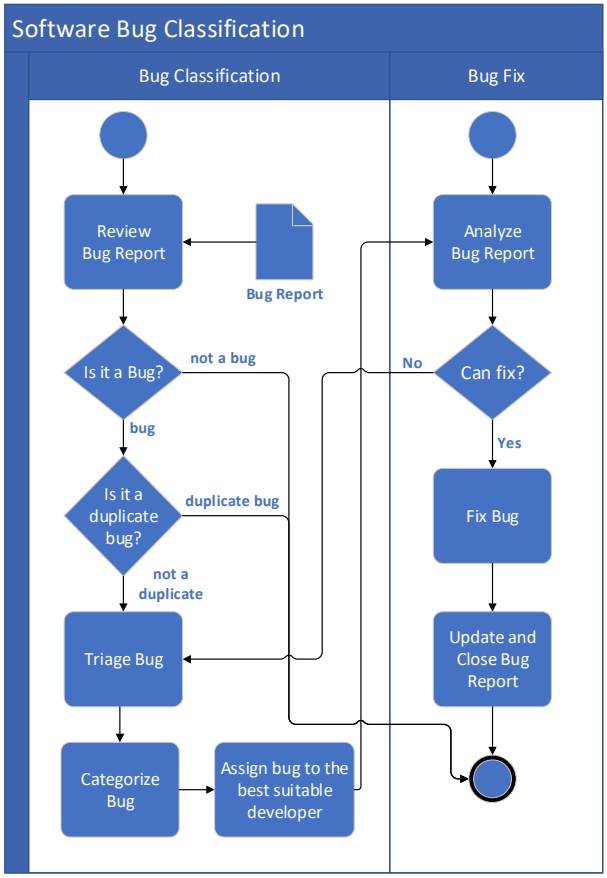

Fig. 1. Software Bug Classification Process.

IV. Artificial Intelligence-based Research on Software Engineering

Four main categories of artificial intelligence (AI) applications for software bug classification are the following: (1) Detecting whether a bug is a real bug or a new feature is requested, (2) Detecting whether a bug is a duplicate bug, (3) Bug triaging, and (4) Software Bug Report Classification. Apart from the above items, several other AI-based applications are used in software verification and validation fields, such as predicting software defects and predicting bug fixing efforts. But, these are out of the scope of this work. So, in the next sections, we focus on bug report classification and give brief information about AI applications on software verification and validation.

A. Software Bug Datasets

In the literature, most of the studies are performed using open source projects' bug datasets such as Eclipse, Mozilla, JBoss, and several Apache projects. However, several types of research are performed on bug datasets and their data quality. For example, Sánchez et al. [12] created a TANDEM dataset to classify the performance bugs. Performance bugs are errors in software that makes the software inefficient or slow. This dataset categorizes performance bugs into five subsets according to their origin.

Wu et al. [13] worked on five publicly available datasets used for classifying bug reports as security or non-security bug reports. They manually re-labeled the samples of the datasets and found that there were many mislabeled sentences. Furthermore, they observed that the classification performance of the ML algorithms with the re-labeled datasets was better than the same algorithms with the original datasets.

B. Bug Type Classification Schemas

Our work focused on the problem of categorizing software bugs into predefined bug classification schemas. Researchers offered several defect classification schemas.

Beizer [1] offered a well-known scheme in which he indicated that various sources affect the taxonomy. Hence over a hundred software fault categories, including ten main categories, are provided in his work.

Orthogonal Defect Classification (ODC) is another widely used defect classification schema by Chillarege et al. [2]. ODC evaluates defects at two specific times: when defects are opened and closed. Three attributes are captured when defects are opened: activity, trigger, and impact. On the other hand, five attributes are captured at closing time: target, defect type, qualifier, source, and age. Authors offer to reduce the number of defects by classifying them in terms of the involved program structure. This methodology aims to enable feedback to software developers and was applied to over 50 IBM software projects.

Freimut et al. [14] proposed an approach as an alternative to using ODC schemes since they evaluated that it is hard to transfer existing schemes to ODC. The authors present a new approach to introduce, describe, and confirm adopted defect categorization schemas for industrial use in their work. The authors stated their main goal of combining software engineering and domain know-how. To this end, the authors defined defects by their three properties as origin, mode, and type. 'Origin' is defined as the activity in which the defect was introduced, 'mode' details the scenarios leading to the defect, and the 'type' characterizes the bug's location.

Seaman et al. [3] offer software bug classification schemes using historical data to guide future projects. The authors provide three types of defects which are "requirements inspection", "design and source code inspection", and "test plan inspection".

Ploski et al. [15] compared various software fault categorization schemas by performing a literature survey. The authors indicated that the generalization of quantitative observations concerning software faults among projects is still open and concluded that the distribution of software faults tightly depends on project-specific factors.

C. Bug Report Quality

Several studies have focused on automated defect classification proposing a solution to the various issues in this area. A software bug report is the vital input of the bug classification process; hence, the bug report quality is a dominant factor affecting the whole classification process. Therefore, several studies have focused on the quality of bug reports quality. For example, Bettenburg et al. [16] and

Zimmermann et al. [17] reported their survey results to find the vital information that shall be included in software defect descriptions. For this purpose, a tool was developed to measure the quality of the bug reports and guide bug reporters. So, with the use of this tool, the reporters are guided to provide helpful information to improve the quality of the software bug reports.

D. Duplicate Bug Reports

After reporting the bug, the bug reports need to be checked to see if the same bug has already been entered into the bug report database. However, Bettenburg et al. [16] and Zimmermann et al. [17] specified that reproduced/duplicate bug reports are harmless, contrary to common belief. However, Sun et al. [18] denoted that the duplicate definitions might cause extra maintenance costs.

E. Detecting Whether a Report is a Bug or Not

The next step in the process in Figure 1 is checking whether the bug report includes a real bug definition or the definition of a new feature instead of a bug encountered.

Terdchanakul et al. [19] provided an N-gram and inverse document frequency (IDF) solution for this problem. The authors claimed they achieved F1 scores between 62% and 88% for this task.

Another research in this task belongs to Antoniol et al. [20]. The authors automatically classified defect definitions in free text format with traditional data mining algorithms. As a result, the authors claimed that more than fifty percent of errors in defect report classification are not related to software bugs. Further, the developed ML-based categorization approach distinguishes defects from other concerns entered into bug reports.

Herzig et al. [21] investigated how misclassification influences bug prediction. The authors concluded that one-third of the bugs are not real bug descriptions.

F. Bug Triaging

Another issue in the bug report classification process given in Figure 1 is bug triaging. Several studies have focused on bug triaging. For example, Čubranic et al. [22] and Neelofar et al. [23] researched how to automatically delegate a software defect in the bug tracking system to the most convenient software developer. In addition, Zhang et al. [24] performed a detailed survey about the research in this domain.

V. Bug Report Classification

This section presents the main research about bug report classification using ML-based algorithms and techniques. Many types of research are performed on software bug classification in the literature. Among these, we select 18 primary studies (PS). In this section, we provide more details about the PS in historical order, and in the next section, we compare these researches based on the techniques and algorithms they used. Further, we provide more details about the datasets used in these works.

Huang et al. [25] proposed an approach named AutoODC for automating the classification process using SVM with ODC schema. The authors declared they have used a manual categorization baseline and using AutoODC, achieving 80.2% accuracy.

Thung et al. [26] used a 500 software defects dataset. The authors first classified and analyzed this dataset manually using ODC. Further, they automatically classified the same dataset using SVM and achieved an accuracy of 77.8 %.

Pingclasai et al. [27] automatically classified software defect reports with a topic modeling algorithm, namely Latent Dirichlet Allocation (LDA). The authors indicated their model's most efficient classifier was Naïve Bayes, whereas LDA improved the F1 score from 65% to 82%.

Another topic modeling algorithm, the nonparametric Hierarchical Dirichlet Process (HDP), was applied by Limsettho et al. [28]. The authors concluded that the HDP performs similarly to LDA, which requires parameter tuning. However, they indicated that, for the used dataset, both LDA and HDP suffer from a lack of data and imbalance problems.

Zhou et al. [29] used a hybrid approach so-called multi-stage classification. They performed automated classification by associating data mining and text mining. Further, by applying their proposed technique to large-scale open-source projects, the authors declared that they improved the F1 measures. The achieved F1 scores were reported as 93.7 for Jboss (+9.1%), 79.5% for Firefox (+6.7%), 85.9% for OpenFOAM (+6.1), 80.2% for Eclipse (+3.5%), and 81.7% for Mozilla (+%3,4),

Xia et al. [30] developed a fUzzy Set based fEature Selection (USES) algorithm to categorize software bugs by inspecting defect reports' natural language descriptions. The authors claimed they increased the best baseline by 12.3% and achieved a 45.3% F1 measure.

Thung et al. [31] proposed an active semi-supervised defect categorization approach to investigate how this algorithm cut down the manual labeling cost of the dataset they used in their previous work. The authors claim that, by incrementally refining the training model, their approach outperforms the baseline work by Jain et al. [32], achieving 62.3% F1 and 71.0% AUC.

Limsettho et al. [33] extended their workshop paper [28], applying the HDP as the topic modeling and using two clustering algorithms: expectation-maximization and X-means. They also worked with a decision tree (DT) and logistic regression (LR) algorithms to compare the clustering algorithms with the supervised algorithms. As expected, the supervised algorithms achieved substantially higher F1 scores than unsupervised ones.

Hernández-Gonzales et al. [34] used the ODC schema [2] in their work and classified the impact of bugs. The authors declared that they had improved the classification performance by using several traditional ML-based classifiers and majority voting techniques.

Catalino et al. [35] declared that researchers investigate bug triaging concepts deeply. Yet, only a few studies exist in the literature to help developers in grasping the category of a reported defect. Therefore, they used various open-source projects to build a bug taxonomy and employ automatic support for labeling software defects concerning their class. As a result, the authors claimed they obtained a 64% F1 measure by implementing their taxonomy and automatic classification process.

Jain et al. [36] created and worked on a dataset of bugs in Web apps. The dataset consists of bug reports from the qaManager, bitWeaver, and WebCalendar apps and reviews on the Google Store of the Dineout: Reserve a Table, and

Wynk Music apps. First, they extracted the features with the term frequency-inverse document frequency (TF-IDF) to process with the particle swarm optimization (PSO) algorithm. Finally, the bugs were classified with five algorithms: NB, DT, SVM, KNN, and MLP. The best accuracy score in the study was 93.35%, achieved by the DT classifier.

Peters et al. [37] developed the FARSEC (**F**iltering **A**nd **R**anking methods to reduce the mislabeling of **SEC**urity bug reports) framework for classifying bug reports as security or non-security bug reports. In the FARSEC framework, the stop words and unwanted terms in bug reports were removed. Unwanted terms were words with punctuation or non-alphanumeric characters. Then, a TF-IDF matrix was created where the features are the most common 100 words. Almost all bug reports in the utilized datasets were non-security bug reports. Also, the authors observed that many non-security bug reports in the datasets were mislabeled. Thus, FARSEC detected the mislabeled samples by Graham's Bayesian filter and removed these samples. The bugs in the dataset were classified with NB, LR, RF, MLP, and KNN algorithms.

Lopes et al. [38] used ODC schema [2] to evaluate several ML-based algorithms (KNN, SVM, NB, Nearest Centroid, RF, and RNN) for automatic bug categorization utilizing ODC schema on unstructured textual bug reports. The authors declared that their experiments demonstrate the challenges of automatically categorizing particular ODC attributes solely using bug reports. Hence, they suggested improving classification accuracy using larger datasets.

Kumar et al. [39] have trained an LSTM model for bug report classification. They used ODC defect types (Chillarege et al., 1992) as the classes. They achieved a precision of 59.4% and a recall of 57.2%. With these scores, their model outperformed the TF-IDF model introduced by Lopes et al. [38], who also published the dataset.

Zheng et al. [40] worked on a highly imbalanced dataset with two classes. The dataset had 128,946 bug reports in total, of which 127,536 reports belong to the majority class. Six class rebalancing techniques were used to solve the class imbalance problem: adaptive synthetic, borderline -SMOTE, near miss, condensed nearest neighbor, Rose, and Mahakil. Then, they trained LR, multinomial NB, MLP, SVM, and RF models for bug report classification. It can be observed from the results that classification performance improves vastly when class rebalancing techniques are used. For example, using Rose and RF together achieved an F1 score of 83.0% on the Chromium dataset, whereas the RF model achieved an F1 score of 67.0% on the same dataset.

Ahmed et al. [41] created a framework named as CaPBug. This framework categorizes bug reports into bug types. Bug reports from Mozilla and Eclipse were used by manual labeling. The reports included both textual and categorical features. Both using only the textual features give the best results. Ahmed et al. used SMOTE to overcome the class imbalance problem. This technique over-samples majority classes and under-samples minority classes. After applying SMOTE, they achieved an accuracy score of 88.78% with the RF classifier.

Köksal et al. [42] performed a bug classification on a commercial dataset using several traditional ML-based algorithms (NB, KNN, LR, DT, RF, and SVM with four different kernels). The authors stated that small datasets are unsuitable for training DL algorithms such as CNN and RNN. They also used FastText [8] word embeddings and BERT [9] models but obtained the best results using SVM in their commercial dataset.

Kim et al. [43] utilized transfer learning for bug report classification. Firstly, they trained a CNN model with source code files. During the first training session, the model's input was the file names and file contents as texts. Then, they re-trained this model with bug reports. During this training session, the model's input was the summary of the bug reports (used as a replacement of the file names) and the contents of the bug reports (used as a replacement of the file contents). In this study, two classes of bugs were defined. A macro-F1 score of 83.9% was achieved.

VI. Comparison of Related Works

In this section, we compare bug report classification works mentioned in the previous sections. We selected 19 primary studies (PS) from the related work section. In Table 1, we have listed the datasets' properties used in these PS.

TABLE I. DataSets used in Primary Studies.

Primary Studies	Ref.	Year	Dataset (DS) Name	DS Type	No. of Bugs
Huang et al.	[25]	2011	Proprietary / commercial dataset	PD	403
Thung et al.	[26]	2012	Mahout, Lucene, OpenNLP	OS	500
Pingclasai et al.	[27]	2013	HTTP Client, Jackrabbit, Lucene	OS	1,940
Limsettho et al.	[28]	2014	HTTP Client, Jackrabbit, Lucene	OS	2,718
Zhou et al.	[29]	2014	Mozilla, Eclipse, JBoss, Firefox, OpenFOAM	OS	3,220
Xia et al.	[30]	2014	Apache HTTPD, AXIS, Linux, MySQL,	OS	809
Thung et al.	[31]	2015	Mahout, Lucene, OpenNLP	OS	500
Limsettho et al.	[33]	2016	HTTP Client, Jackrabbit, Lucene	OS	4,710
Hernández et al.	[34]	2018	Compendium, Mozilla	OS	1,444
Catolino et al.	[35]	2019	Mozilla, Apache, Eclipse	OS	1,280
Jain et al.	[36]	2019	qaManager, bitWeaver, WebCalendar, Dineout: Reserve a Table, and Wynk Music	OS	4,577
Peters et al.	[37]	2019	Chromium, Wicket, Ambari, Camel, Derby	OS	45,940
Lopes et al.	[38]	2020	MongoDB, Cassandra, HBase	OS	4,096
Kumar et al.	[39]	2021	MongoDB, Cassandra, Hbase	OS	4,096
Zheng et al.	[40]	2021	Ambari, Camel, Derby, Wicket, Chromium, OpenStack	OS	128,946
Ahmed et al.	[41]	2021	Mozilla and Eclipse	OS	2,138
Köksal et al.	[42]	2021	Proprietary / commercial dataset	PD	504
Kim et al.	[43]	2022	Bench4BL	OS	4,059

As can be seen from Table 1, almost all researchers use open source datasets, and only two papers used proprietary/commercial datasets.

Table 2 provides the techniques, schemas, and algorithms used in the selected 18 primary studies. We present this table in the same order as Table 1.

TABLE II. COMPARISON OF AUTOMATIC BUG CLASSIFICATION RESULTS IN THE RELATED WORK.

Paper	Techniques / Schemas Used	Automated Classification / Clustering Algorithms
Huang et al.	Automating ODC -based classification	SVM
Thung et al.	Text Mining using three super categories of ODC defect types	C4.5, NB, SVM, LR
Pingclasai et al.	LDA topic modeling	NB, LR, Alternating DT
Limsettho et al.	HDP topic modeling	NB, LR, Alternating DT
Zhou et al.	Data/text mining and multi-stage classification	Multinomial NB, LR, Alternating DT
Xia et al.	Fuzzy-based text selection and mining, natural language descriptions, and developed USES algorithm	NB, Multinomial NB, LR, SVM, RBF Network, USES
Thung et al.	Hybrid ML approach with clustering and active learning	K-means clustering and SVM
Limsettho et al.	HDP topic modeling	Expectation-Maximization, X-means, DT, LR
Hernández-González et al.	Bayesian network classifiers, K-means clustering, and Expectation-Maximization (EM) strategy with majority voting and ODC defect types	NB, Tree Augmented NB, K-Dependence Bayesian Network Classifier
Catolino et al.	Taxonomy development and classification using the defined taxonomy, using TF-IDF and word embeddings (Word2Vec, Doc2Vec)	NB, LR, SVM, and RF
Jain et al.	TF-IDF-based feature extraction and PSO algorithm	NB, DT, SVM, MLP, KNN
Peters et al.	Detection of mislabeled samples with Graham's Bayesian filter.	NB, LR, RF, MLP, KNN
Lopes et al.	Automatic classification using all ODC defect types	NB, Nearest Centroid, SVM, KNN, RF, RNN
Kumar et al.	ODC defect types	LSTM
Zheng et al.	Class rebalancing methods (adaptive synthetic, borderline-SMOTE, near miss, condensed nearest neighbor, Rose, and Mahakil)	LR, MNB, MLP, SVM, RF
Ahmed et al.	SMOTE as the class rebalancing method	NB, DT, RF, LR
Köksal et al.	ML, NLP, information retrieval, and with Seaman's defect types	NB, SVM, KNN, LR, DT, RF
Kim et al.	Transfer learning from a source code classification model	CNN

By comparing these works, we aim to underline similarities of the utilized processes while indicating the diverse methodologies. We present the properties of the datasets used in the selected papers in Table 1. Further, we provide the techniques and the bug classification schema used (if they exist) in the selected studies in Table 2, denoting the names of the classification/clustering algorithms used.

VII. DISCUSSION AND CONCLUSION

Automated software bug classification is a widely used process in software verification and validation. ML is widely used in automated software bug classification. Automated bug classification is time and effort-saving and might be more robust than manual bug classification. Hence, this paper focuses on the ML-based automatic classification of software bug reports and presents important research in this domain.

First, we presented the main steps of the software bug classification process, indicating the fundamental concerns. Then, we provided several types of research on the sub-processes defined, including bug report quality and defect classification schemes. Finally, we provide the details of 18 PS.

Our investigation of selected PS stated that categorization of software bugs becomes more difficult if we deal with unstructured bug reports. We have seen that most primary studies used traditional ML algorithms as classifiers in their research. DL approaches have been widely used in software bug classification in recent years. Further, we presented various approaches and particular techniques used in this domain. Moreover, we have provided the names and types of the datasets used in primary studies. Finally, we have seen that most of the datasets used are open-source. Hence, we conclude that there is room for further research to be executed using commercial datasets.

We plan to prepare a more detailed survey for bug report classification highlighting the main obstacles and presenting the proposed solutions as future work.

REFERENCES

[1] B. Beizer, *Software testing techniques*. Van Nostrand Reinhold, 1990.

[2] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, B. K. Ray, and D. S. Moebus, "Orthogonal Defect Classification—A Concept for In-Process Measurements," IEEE Trans. Softw. Eng., vol. 18, no. 11, pp. 943–956, 1992.

[3] C. Seaman et al., "Defect categorization: Making use of a decade of widely varying historical data," in ESEM'08: Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 149–157.

[4] Ö. Köksal, "Classification of Text Data in Healthcare Systems – A Comparative Study," in Machine Learning and Deep Learning in Efficacy Improvement of Healthcare Systems, CRC Press, 2022, pp. 333–352.

[5] Ö. Koksal, "Artificial Intelligence-Based Categorization of Healthcare Text," in Internet of Things and Data Mining for Modern Engineering and Healthcare Applications, Chapman and Hall/CRC, 2022, pp. 89–106.

[6] B. Alshemali and J. Kalita, "Improving the Reliability of Deep Neural Networks in NLP: A Review," Knowledge-Based Syst., vol. 191, p. 105210, Mar. 2020.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in 1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings, 2013.

[8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," Trans. Assoc. Comput. Linguist., vol. 5, pp. 135–146, Dec. 2017.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.

[10] K. Clark, M.-T. Luong, Q. V Le, and C. D. Manning, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators," ArXiv, vol. abs/2003.1, 2020.

[11] OMG, “BPMN Specification - Business Process Model and Notation.” [Online]. Available: http://www.bpmn.org/. [Accessed: 08-Jun-2020].

[12] A. B. Sanchez, P. Delgado-Perez, I. Medina-Bulo, and S. Segura, “TANDEM: A Taxonomy and a Dataset of Real-World Performance Bugs,” IEEE Access, vol. 8, pp. 107214–107228, 2020.

[13] X. Wu, W. Zheng, X. Xia, and D. Lo, “Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction,” IEEE Trans. Softw. Eng., Jul. 2021.

[14] B. Freimut, C. Denger, and M. Ketterer, “An industrial case study of implementing and validating defect classification for process improvement and quality management,” in Proceedings - International Software Metrics Symposium, 2005, vol. 2005, pp. 165–174.

[15] J. Ploski, M. Rohr, P. Schwenkenberg, and W. Hasselbring, “Research issues in software fault categorization,” ACM SIGSOFT Softw. Eng. Notes, vol. 32, no. 6, p. 6, Nov. 2007.

[16] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?,” in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2008, pp. 308–318.

[17] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, “What makes a good bug report?,” IEEE Trans. Softw. Eng., vol. 36, no. 5, pp. 618–643, 2010.

[18] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, “Towards more accurate retrieval of duplicate bug reports,” in 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings, 2011, pp. 253–262.

[19] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, “Bug or not? Bug Report classification using N-gram IDF,” in Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, 2017, pp. 534–538.

[20] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y. G. Guéhéneuc, “Is it a bug or an enhancement? A text-based approach to classify change requests,” in Proceedings of the 2008 Conference of the Center for Advanced Studies, CASCON’08, 2008.

[21] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: How misclassification impacts bug prediction,” in Proceedings - International Conference on Software Engineering, 2013, pp. 392–401.

[22] D. Čubranić and M. Gail, “Automatic bug triage using text categorization,” SEKE 2004 Proc. Sixt. Int. Conf. Softw. Eng. Knowl. Eng., pp. 92--97, 2004.

[23] Neelofar, M. Y. Javed, and H. Mohsin, “An automated approach for software bug classification,” in Proceedings - 2012 6th International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012, 2012, pp. 414–419.

[24] T. Zhang, H. Jiang, X. Luo, and A. T. S. Chan, “A Literature Review of Research in Bug Resolution: Tasks, Challenges, and Future Directions,” Comput. J., vol. 59, no. 5, pp. 741–773, May 2016.

[25] L. G. Huang, V. Ng, I. Persing, R. Geng, X. Bai, and J. Tian, “AutoODC: Automated generation of Orthogonal Defect Classifications,” in 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings, 2011, pp. 412–415.

[26] F. Thung, D. Lo, L. Jiang, X. B. D. Le, and D. Lo, “Automatic defect categorization,” in Proceedings - Working Conference on Reverse Engineering, WCRE, 2012, vol. 2015-August, pp. 205–214.

[27] N. Pingclasai, H. Hata, and K. I. Matsumoto, “Classifying bug reports to bugs and other requests using topic modeling,” in Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2013, vol. 2, pp. 13–18.

[28] N. Limsettho, H. Hata, and K. I. Matsumoto, “Comparing hierarchical dirichlet process with latent dirichlet allocation in bug report multiclass classification,” in 2014 IEEE/ACIS 15th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2014 - Proceedings, 2014.

[29] Y. Zhou, Y. Tong, R. Gu, and H. Gall, “Combining text mining and data mining for bug report classification,” in Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014, 2014, pp. 311–320.

[30] X. Xia, D. Lo, X. Wang, and B. Zhou, “Automatic defect categorization based on fault triggering conditions,” in Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS, 2014, pp. 39–48.

[31] F. Thung, X. B. D. Le, and D. Lo, “Active Semi-supervised Defect Categorization,” in IEEE International Conference on Program Comprehension, 2015, vol. 2015-August, pp. 60–70.

[32] P. Jain and A. Kapoor, “Active learning for large multi-class problems,” 2009, pp. 762–769.

[33] N. Limsettho, H. Hata, A. Monden, and K. Matsumoto, “Unsupervised Bug Report Categorization Using Clustering and Labeling Algorithm,” http://dx.doi.org/10.1142/S0218194016500352, vol. 26, no. 7, pp. 1027–1053, Sep. 2016.

[34] J. Hernández-González, D. Rodriguez, I. Inza, R. Harrison, and J. A. Lozano, “Learning to classify software defects from crowds: A novel approach,” Appl. Soft Comput. J., vol. 62, pp. 579–591, Jan. 2018.

[35] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, “Not all bugs are the same: Understanding, characterizing, and classifying bug types,” J. Syst. Softw., vol. 152, pp. 165–181, Jun. 2019.

[36] D. K. Jain, A. Kumar, S. R. Sangwan, G. N. Nguyen, and P. Tiwari, “A Particle Swarm Optimized Learning Model of Fault Classification in Web-Apps,” IEEE Access, vol. 7, pp. 18480–18489, 2019.

[37] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, “Text Filtering and Ranking for Security Bug Report Prediction,” IEEE Trans. Softw. Eng., vol. 45, no. 6, pp. 615–631, Jun. 2019.

[38] F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro, and J. Bernardino, “Automating orthogonal defect classification using machine learning algorithms,” Futur. Gener. Comput. Syst., vol. 102, pp. 932–947, Jan. 2020.

[39] S. Kumar, M. Sharma, V. B. Singh, and S. K. Muttoo, “Bug Report Classification into Orthogonal Defect Classification Defect Type using Long Short Term Memory,” Proc. - 2021 3rd Int. Conf. Adv. Comput. Commun. Control Networking, ICAC3N 2021, pp. 285–287, 2021.

[40] W. Zheng, Y. Xun, X. Wu, Z. Deng, X. Chen, and Y. Sui, “A Comparative Study of Class Rebalancing Methods for Security Bug Report Classification,” IEEE Trans. Reliab., vol. 70, no. 4, pp. 1658–1670, Dec. 2021.

[41] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, “Capbug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms,” IEEE Access, vol. 9, pp. 50496–50512, 2021.

[42] Ö. Köksal and B. Tekinerdogan, “Automated Classification of Unstructured Bilingual Software Bug Reports: An Industrial Case Study Research,” Appl. Sci., vol. 12, no. 1, 2022.

[43] M. Kim, Y. Kim, and E. Lee, “Deep Learning-based Production and Test Bug Report Classification using Source Files,” Proc. - Int. Conf. Softw. Eng., pp. 343–344, 2022.