International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Machine Learning Based Predictive Analysis of Software Bug Severity and Priority

[1]Kamna Vaid, [2]Prof. Udayan Ghose

Submitted: 12/12/2023 **Revised**: 28/01/2024 **Accepted**: 03/02/2024

Abstract: Software fault prediction is a vital and helpful technique for boosting the quality and dependability of software. There exists the prospective to enhance project management by proactively estimating prospective release delays and implementing cost-effective measures to boost software quality. This can be achieved by forecasting the components within a sizable software system that are most likely to exhibit a significant number of flaws in subsequent releases. However, creating reliable fault prediction models is a difficult task. This study's primary goal is to carry out an investigation into the predictive analysis of software development frameworks with regard to software bug attributes: severity and priority. The machine learning method utilized in this study was implemented by using the Python programming platform. The implementation of this study makes use of methods from AI, along with data mining, and Machine Learning, along with statistical algorithms, and also modelling. Prediction models can be of assistance in maximizing all of the resources needed for the research. Random Forest (RF) Classifier and Support Vector Machine (SVM) are two techniques used in machine learning model training to determine the severity and urgency of the problem. Per the findings of the study, The RF Priority Model provides a detailed outlook of the model's predicted performance across different priority levels with an accuracy rate of 0.87. This investigation assists developers discover faults based on existing software metrics using data mining techniques, which eventually will lead to an improvement in software quality and a decrease in the cost of developing software during both the development phase and the maintenance phase.

Keywords: *Predictive Analysis; Predictive Models; Software Bugs; Priority; Severity; SVM; Random Forest Classifier.*

1. Introduction

The "software defect prediction" (SDP) refers to the process of proactively identifying problematic aspects of a software system. The software development life cycle includes a number of various testing methodologies, one of the most essential of which is cloud-based solutions (Ali et al., 2022). This stage is also an extremely time-consuming stage (Shatnawi et al., 2022). Early prediction and detection of problematic parts should typically demand quick debugging, the nature of which is determined by the severity degree of the defect or defects that have been found. In addition, the phase of the software development process known as the gathering of software requirements is an important early stage. As a result, testing the software while it is being developed is given a high priority in order to guarantee that it complies with the requirements specifications (Akmel et al., 2017). A software defect refers to a flaw, mistake, or error inside a software, resulting in an unexpected production along with an unintended behavioural effect that contradicts the quality objectives of software engineers along with the anticipated outcomes of end-users (Ali et al., 2019; Ali et al., 2022; Olaleye et al., 2021).

[1]*Research Scholar, University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University, Dwarka, New Delhi, India*
Email ID: kamnavaid@gmail.com
[2]*University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University, Dwarka, New Delhi, India*

Due to a lack of software testing, software defects are the most significant features that come with every release of a software product (Zhang et al., 2018). Developers encourage users to report defects via issue tracking systems like Bugzilla, along with Mantis, along with Google Code Issue Tracker, along with GitHub Issue Tracker, JIRA, and many others to prevent the existence of similar bugs in upcoming releases or new software products. By reporting defects, which is a common practice in the software maintenance process, users assist developers.

In cloud platform data, bugs are typically encountered. Problems with the cloud's logic (29%), load (4%), space (4%), error-handling (18%), along with optimization (15%), along with configuration (14%), along with data-race (12%), and hang (4%), are only some of the software bugs that can occur. The fact that nearly any kind of bug can result in practically any kind of consequence, including failed operations, along with performance degradation, component outages, data loss, along with staleness, along with corruption, makes the problem significantly worse. The purpose of this study is to utilise software prediction techniques to assess the severity and priority of faults in software as well as to prevent flaws from entering the system. The following section elaborates on the past literature related to this field.

2. Literature Review

In the studies conducted, Malgonde and Chari (2019) utilized computer experiments to showcase the superior performance of the ensemble-based technique compared to previous ensemble-based benchmarking approaches. The research utilized an optimization model to enhance the sprint planning for two projects based on the dataset. This demonstrated the practical usability of the prediction model and the ensemble-based technique.

In the study, Shetty et al. (2020) showcased the successful deployment of a rule-based expert system for fault remediation. Additionally, they proposed an analytical model so as to accurately evaluate the efficiency of remediation techniques for various fault types. Related to a reactive fault controlling system, the results of an experiment using a specifically designed prototype demonstrate enhanced availability with reduced overhead. According to Ahmed et al. (2021), the framework is created utilizing NLP along with supervised machine learning methods. By examining more than 2000 bug complaints obtained from the Mozilla along with Eclipse sources. Four classification algorithms, which are Naive Bayes, along with RF, along with Decision Tree, along with Logistic Regression—were employed to categorize along with prioritize problem reports. Also, the study showcased the utilization of the CaPBug framework, which utilized a RF classifier together with a textual feature to predict the category. Also, the framework attained a precision rate of 88.78%. Similarly, the CaPBug framework attained a precision rate of 90.43% in evaluating the importance of bug reports. The problem of class imbalance in priority classes has been addressed by applying SMOTE. Distinguish resilience from the concept of reliability (SMOTE).

Pachouly et al. (2022) emphasized the necessity of conducting an all-inclusive exploration of software defect prediction to investigate datasets, along with data validation procedures, defect detection, along with prediction tactics along with tools. Also, the findings of the literature research revealed that the conventional datasets possess a limited number of labels, thereby offering less information on the challenges at hand. The study proposed a methodology for creating a software prediction dataset that includes a sufficient number of attributes. It also employed statistical data validation approaches to accurately classify software concerns into many categories.

According to Alsaedi et al. (2023), a new prediction model was proposed to evaluate BRs along with forecast the characteristics of bugs. The proposed model combines NLP and ML techniques to form an ensemble machine learning method. The simulation outcomes showcased that the proposed model, while utilizing text augmentation, achieved an accuracy of 96.72%, but without text augmentation, it achieved an accuracy of 90.42%. These results indicate that the recommended model outperforms the bulk of existing models in terms of accuracy.

Software defect prediction is a significant and helpful approach for boosting the quality and dependability of software, according to previous literature. There exists the potential to enhance project management practices by implementing early estimation techniques to identify potential release complications, as well as employing cost-effective strategies to guide corrective actions aimed at improving software quality (Alsghaier et al. (2021). This can be achieved by utilising predictive models to determine the components within a large software system that are most likely to exhibit a significant number of flaws in subsequent releases. However, creating reliable fault prediction models happens to be a tough issue, and numerous methods are put forth inside the collected works.

Therefore, the main goal of this study is to undertake a research project on the predictive analysis of software development frameworks in relation to software problem traits, severity, and priority.

3. Methodology

One cannot completely eradicate bugs, fixes, patches, and other software-related issues because each one has a severity and priority level. This study was carried out utilizing the Python programming language. The implementation strategy for the prediction model will be as follows.

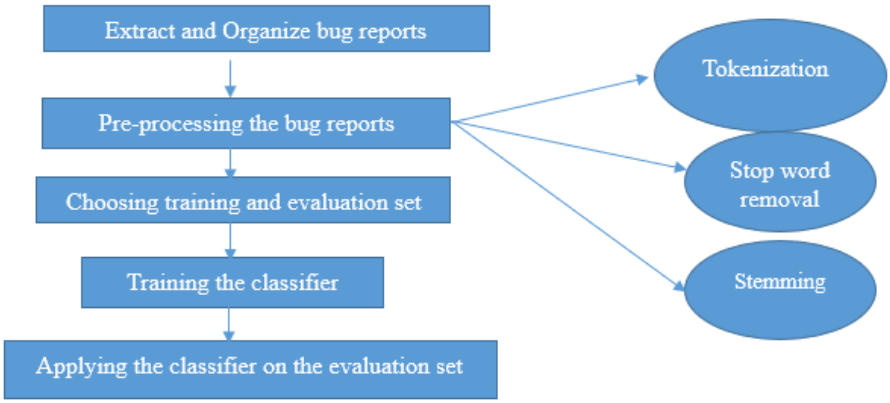

Fig 1: Proposed Methodology of this study

DATASET

The dataset utilized in this study, focuses on the predictive analysis of software development frameworks in relation to software bug attributes: severity, and priority. The dataset for agile software development, was created using five different open-source programs: Apache, JIRA, JBoss, MongoDB, and Spring. Bug reports comprising of multiple attributes have been extracted for all the five open source softwares, obtained from the website: https://bz.apache.org/.

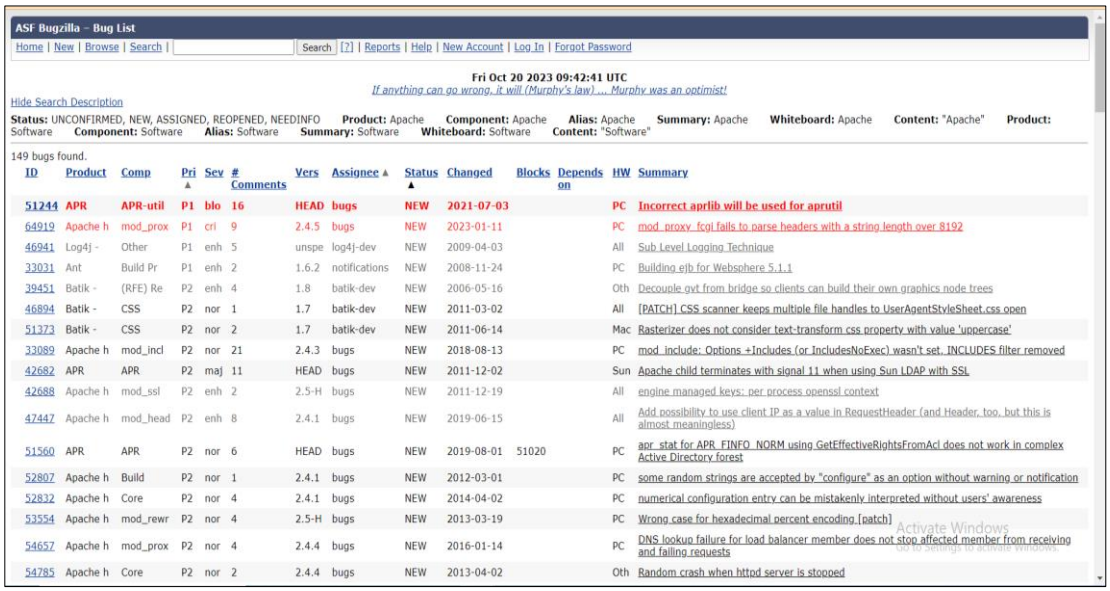

Fig 2: Dataset used for this study

LIBRARIES:

Pandas library is utilized for data analysis along with processing, while Matplotlib and Seaborn are used for data visualization. These libraries enable the creation of useful charts and graphs. Additionally, scikit-learn modules like train_test_split, LabelEncoder, RandomForestClassifier, SVC, and classification_report are essential for data preprocessing, model training, and performance evaluation.

DATA PRE-PROCESSING AND FEATURE ENGINEERING:

The objective of this code is to organize bug data for machine learning analysis. First, it replaces missing information in the "Assignee" section with "Unknown." Datetime objects are created from "Changed" column data. Feature engineering begins with the 'Changed_Year' and 'Changed_Month' columns, which pull year and month from the 'Changed' date. Encoding categorical information including product, component, assignee, status, version, depends on, hardware, summary, blocks, severity, and priority with the LabelEncoder improves machine learning algorithms. We can also encode blocks, severity, and priority. This stage formats the data for model training along with evaluation.

SPLITTING THE DATA AND MODEL TRAINING:

The code partitions the dataset into training along with testing sets and trains Random Forest models to predict issue severity and priority, completing an important machine learning pipeline stage. Scikit-learn's train_test_split function splits data into features (X) and severity and priority target variables (y_severity and

y_priority). Stratified splits enable consistency in severity and priority labels across training and testing sets. Two Random Forest classifiers were trained using the training set, with each classifier consisting of 100 decision trees. Classifiers acquire knowledge of patterns and correlations within a given dataset in order to make predictions for instances that have not yet been included in the testing set. The fundamental basis has been established. These metrics and images are intended to facilitate model performance evaluations at a later stage.

TRAINING MACHINE LEARNING MODELS:

Two methods are employed in machine learning model training to estimate problem severity and priority: Random Forest Classifier and SVM. The severity and priority forecasts for each algorithm are trained separately. The ensemble structure of the RandomForestClassifier provides robustness against overfitting and is implemented with 100 decision trees. Training the model involves using the feature matrix X_train and severity labels y_severity_train for severity prediction (severity_model). To train the priority model, the feature matrix X_train and priority labels y_priority_train are used with the same parameters.

4. Results and Discussions

Predictions for bug severity are produced using both the Random Forest model (severity_model) and the SVM model (svm_severity_model). The classification reports are printed in order to assess the models' efficacy. Each class's classification report includes detailed summaries of a number of parameters, including as precision, along with recall, along with F1-score, along with support. Separate reports for Random Forest along with SVM models are presented for problem severity and prioritization. These metrics highlight areas where the models can improve their ability to reliably classify bug severity and importance. This approach makes it easier to understand the effectiveness of the models. The severity model report is given in table below. Classification reports for every class include detailed summaries of multiple parameters, including as precision, along with recall, along with F1-score, along with support. Metrics for problem severity and priority highlight areas for improvement and potential directions for the models' precise bug severity and priority classification. This strategy makes understanding the effectiveness of the models easier.

Table 1: Precision, along with Recall, along with F1- Score for various models

RANDOM FOREST SEVERITY MODEL REPORT:				
CLASS	**PRECISION**	**RECALL**	**F1-SCORE**	**SUPPORT**
0	0.00	0.00	0.00	0
1	0.00	0.00	0.00	2
2	0.14	0.50	0.22	2
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	1
5	0.85	0.74	0.79	3
Accuracy			0.60	30
Macro avg	0.17	0.21	0.17	30
Weighted avg	0.66	0.60	0.62	30
RANDOM FOREST PRIORITY MODEL REPORT:				
CLASS	**PRECISION**	**RECALL**	**F1-SCORE**	**SUPPORT**
0	0.00	0.00	0.00	1
1	0.86	1.00	0.92	24
2	1.00	0.50	0.67	4
4	0.00	0.00	0.00	1
Accuracy			0.87	30
Macro avg	0.46	0.38	0.40	30

Weighted avg	0.82	0.87	0.83	30
SVM SEVERITY MODEL REPORT:				
CLASS	**PRECISION**	**RECALL**	**F1-SCORE**	**SUPPORT**
1	0.00	0.00	0.00	2
2	0.14	0.50	0.22	2
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	1
5	0.86	0.78	0.82	23
Accuracy			0.63	30
Macro avg	0.20	0.26	0.21	30
Weighted avg	0.67	0.63	0.64	30
SVM PRIORITY MODEL REPORT:				
CLASS	**PRECISION**	**RECALL**	**F1-SCORE**	**SUPPORT**
0	0.00	0.00	0.00	
1	0.92	0.96	0.94	
2	0.60	0.75	0.67	
4	0.00	0.00	0.00	
Accuracy			0.87	30
Macro avg	0.38	0.43	0.40	30
Weighted avg	0.82	0.87	0.84	30

Across different severity classes, the Random Forest Severity Model's precision, along with recall, along with F1-score measures differ. The model is remarkably accurate, as evidenced by its F1-score of 0.79 for severity class 5. Overall, the accuracy is 0.60, although class 0 and other severity levels have far lower precision and recall. The precision, along with recall, along with F1-score of the Random Forest Priority Model are more uniformly distributed, with a substantial accuracy of 0.87. The model performs well when predicting class 1 severity but struggles when predicting class 0 and class 2 severity. The reports indicate comparable trends for the Severity and Priority SVM models. The models encounter challenges in accurately forecasting classes other than 5 and 1 in terms of severity and importance. However, it is in these specific classes that the models demonstrate exceptional performance. The distribution of severity is illustrated by using the figure below.

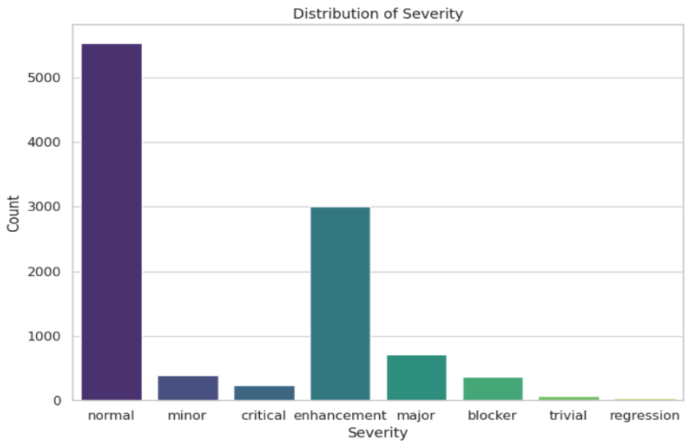

Fig 3: Distribution of Severity

The distribution of priority of bugs can be identified from the figure below.

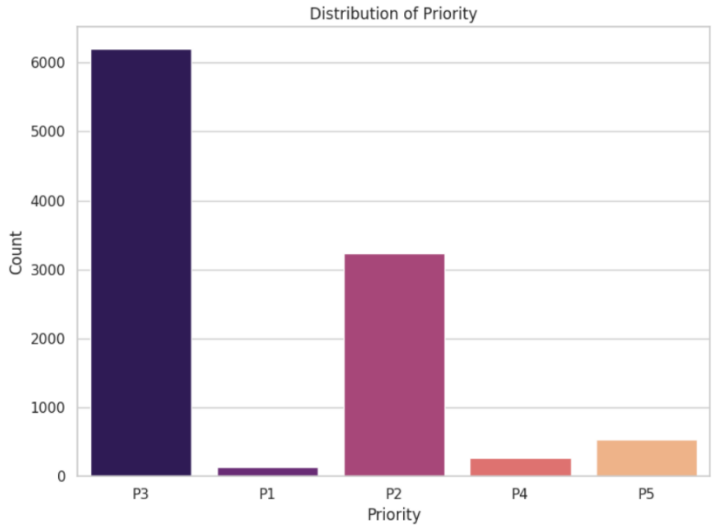

Fig 4: Distribution of priority

The following graphic presents the current status of the severity of the defect as determined by this investigation.

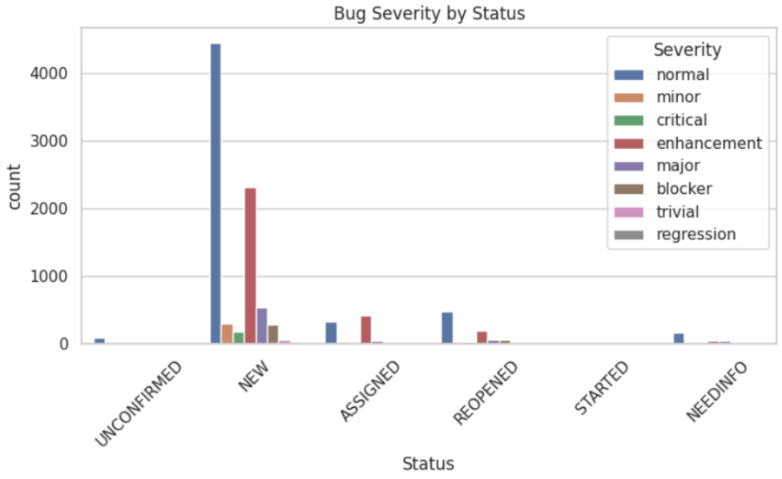

Fig 5: Bug Severity by Status

The following figure presents the severity and priority rankings in their respective distributions.

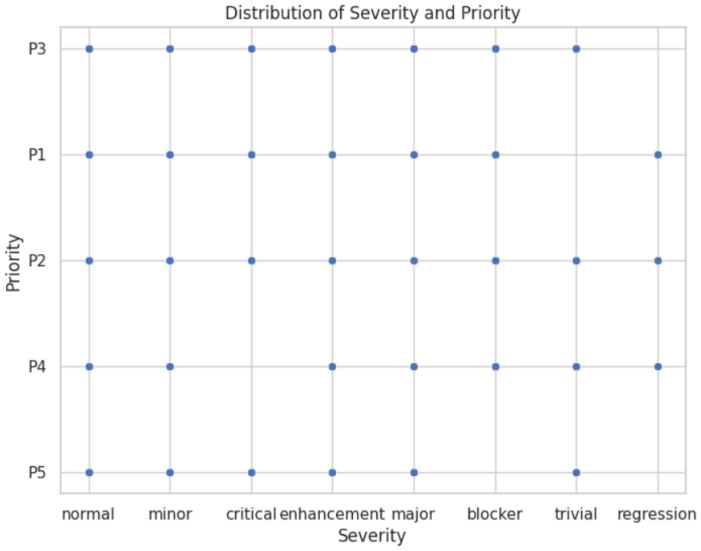

Fig 6: Distribution of Severity and Priority

The histogram of severity levels shows how different severity levels are distributed over the sample. Impact is represented by a problem's severity level, which ranges from 0 to 7. The y-axis of this histogram shows the frequency or count of issues at each severity level, while the x-axis represents each severity level. The distribution of issues with moderate and severe impact is most frequently observed at severity levels 2 and 5, according to the histogram. At severity levels 0, 1, 3, 4, and 7, fewer cases happen. The distribution of priority levels is also shown by the histogram of priority levels, which has a scale from 0 to 4. A problem's priority level indicates how soon it must be resolved. The fact that priority level 1 dominates and levels 0, 2, 3, and 4 are less frequent in this histogram suggests that the majority of the issues require immediate attention. The graph below illustrates the histogram of severity levels.

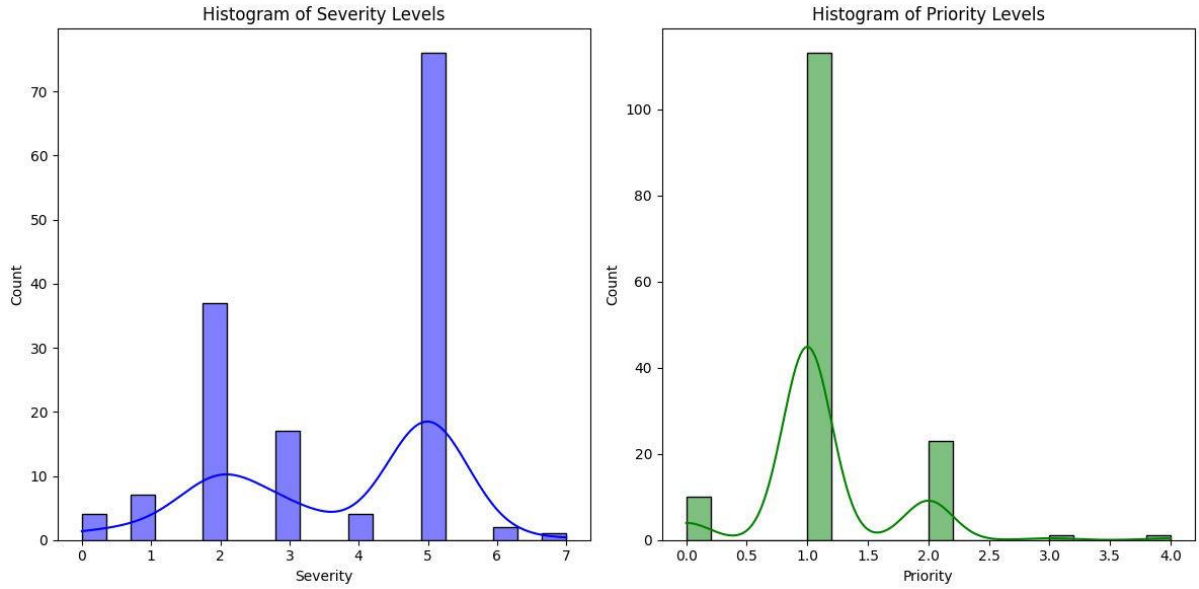

Fig 7: Histogram of Severity and Priority levels.

5. Conclusion

This study aims to predict software development frameworks' bug attributes, severity, and priority. Precision, recall, and F1-score vary by severity class in the Random Forest Severity Model. The model's severity class 5 F1-score of 0.79 shows its accuracy. Class 0 and other severity categories have far poorer precision and recall, but accuracy is 0.60. The Random Forest Priority Model has a robust 0.87 accuracy and more uniform precision, recall, and F1-score. Severity levels are dispersed across the sample in the histogram. Problem severity, ranging from 0 to 7, indicates impact. Given histogram illustrates the frequency or count of issues at each severity level on y-axis along with each severity level on x-axis. Severity levels are dispersed across the sample in the histogram. Problem severity, ranging from 0 to 7, indicates impact. This histogram demonstrates the frequency or count of issues at each severity level on y-axis along with each severity level on x-axis. Finally, the visualization of the Random Forest Priority Model offers a comprehensive view of the model's expected performance across various priority levels. Per the investigation's findings, the inference to be made is that the utilization of a random forest model holds promise for the application of predictive analysis in the context of software development frameworks. Specifically, this approach may be employed to assess software bug attributes, severity, and priority. Hence, it is hypothesized that this investigation acts as a treasured tool for future investigations, providing the research community with a guiding framework to enhance the quality of software defect severity and priority prediction research.

References

[1] Ali, S., Baseer, S., Abbasi, I. A., Alouffi, B., Alosaimi, W., & Huang, J. (2022). Analyzing the interactions among factors affecting cloud adoption for software testing: a two-stage ISM-ANN approach. *Soft Computing*, *26*(16), 8047-8075

[2] Shatnawi, M. Q., & Alazzam, B. (2022). An Assessment of Eclipse Bugs' Priority and Severity Prediction Using Machine Learning. *International Journal of Communication Networks and Information Security*, *14*(1), 62-69.

[3] Akmel, F., Birihanu, E., & Siraj, B. (2017). A literature review study of software defect prediction using machine learning techniques. *Int. J. Emerg. Res. Manag. Technol*, *6*(6), 300-306.

[4] Ali, S., Ullah, N., Abrar, M. F., Majeed, M. F., Umar, M. A., & Huang, J. (2019). Barriers to software outsourcing partnership formation: an exploratory analysis. *IEEE Access*, *7*, 164556-164594.

[5] Ali, S., Nasir, A., Samad, A., Basser, S., & Irshad, A. (2022). An automated approach for the prediction

of the severity level of bug reports using GPT-2. *Security and Communication Networks*, *2022*.

[6] Olaleye, T. O., Arogundade, O. T., Abayomi-Alli, A., & Adesemowo, A. K. (2021). An ensemble predictive analytics of COVID-19 infodemic tweets using bag of words. In *Data science for COVID-19* (pp. 365-380). Academic Press.

[7] Zhang, Y., Chen, Y., Cheung, S. C., Xiong, Y., & Zhang, L. (2018, July). An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis* (pp. 129-140).

[8] Malgonde, O., & Chari, K. (2019). An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering*, *24*, 1017-1055.

[9] Shetty, J., Babu, B. S., & Shobha, G. (2020). Proactive cloud service assurance framework for fault remediation in cloud environment. *International Journal of Electrical & Computer Engineering (2088-8708)*, *10*(1).

[10] Ahmed, H. A., Bawany, N. Z., & Shamsi, J. A. (2021). Capbug-a framework for automatic bug categorization and prioritization using nlp and machine learning algorithms. *IEEE Access*, *9*, 50496-50512.

[11] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. *Engineering Applications of Artificial Intelligence*, *111*, 104773.

[12] Alsaedi, S. A., Noaman, A. Y., Gad-Elrab, A. A., & Eassa, F. E. (2023). Nature-Based Prediction Model of Bug Reports Based on Ensemble Machine Learning Model. *IEEE Access*.

[13] Alsghaier, H., & Akour, M. (2021). Software fault prediction using whale algorithm with genetics algorithm. *Software: Practice and Experience*, *51*(5), 1121-1146.

[14] J.Kim and G. Yang, ‘‘Bug severity prediction algorithm using topic based feature selection and CNN-LSTM algorithm,’’ IEEE Access, vol. 10, pp. 94643–94651, 2022.

[15] https://www.usenix.org/system/files/login/articles/login_aug15_08_gunawi.pdfY.

[16] Tian, D. Lo, X. Xia and C. Sun, "Automated prediction of bug report priority using multi-factor analysis", *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354-1383, 2015.