Nature-Based Bug Report Prediction Using Ensemble Machine Learning with Text Augmentation and XGBoost Extension

Ms. Neha Shireen[1], Mr. Mohammed Fahad Nawaz Khan[2], Mr. Mohammed Aleemuddin Ahmed[3]
Mr. Fazal Ur Rahman[4], Mr. Abdul Fattah[5]
[1]*Assistant Professor, Department. of CSE-AIML, Lords Institute of Engineering and Technology*
[2,3,4,5]*B.E. Students, Department. of CSE-AIML, Lords Institute of Engineering and Technology*

***Abstract*—Software maintenance has evolved into one of the most resource-intensive phases of the software development lifecycle, primarily because of the continuously increasing volume of bug reports generated from large-scale software products. Modern bug repositories such as Mozilla and Eclipse accumulate thousands of reports containing textual descriptions, metadata, severity indicators, and reproduction steps. Manual analysis of these reports for identifying the nature of the bug is time-consuming, error-prone, and operationally expensive. Existing research has focused mainly on severity prediction, priority estimation, duplicate detection, and bug localization, while comparatively limited attention has been given to nature-based multiclass classification.**

This research proposes a text-heavy ensemble machine learning framework for nature-based bug report prediction that combines Natural Language Processing (NLP), TF-IDF feature extraction, text augmentation, multiple baseline classifiers, and an advanced voting ensemble mechanism. The proposed system uses Support Vector Machine, Random Forest, and Logistic Regression as foundational classifiers and combines them through hard and soft voting strategies. To further strengthen predictive capability, an XGBoost extension layer is introduced as the advanced experimental contribution of this work. The model is evaluated on benchmark Eclipse and Mozilla bug repositories containing six bug nature classes: Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code.

Base classifiers such as Support Vector Machine, Random Forest, and Logistic Regression are combined using a voting ensemble classifier to improve prediction robustness. As an extension, XGBoost is incorporated to further enhance classification accuracy. Experimental results show that standalone classifiers achieve 70-77% accuracy, the voting ensemble achieves 89%, and the XGBoost extension achieves the best performance with 92% accuracy.

***Index Terms*—Bug Report Classification, Ensemble Learning, Natural Language Processing, TF-IDF, XGBoost, Software Maintenance, Text Augmentation, Multiclass Classification.**

I. INTRODUCTION

Software systems continue to grow in complexity because of distributed architectures, rapid release cycles, API-driven integrations, and large-scale user adoption. As complexity increases, defects become unavoidable, leading users, testers, and developers to generate large volumes of bug reports in software repositories. A bug report acts as the communication bridge between system failure and developer action. It usually contains a textual summary, detailed description, severity level, reproduction steps, component name, operating environment, and metadata associated with the reporting lifecycle.

The maintenance phase of software engineering depends significantly on the quality and speed of bug report analysis. Traditionally, software teams manually inspect bug reports and determine the category, severity, routing path, and developer assignment. This process becomes impractical when repositories scale to enterprise-level volumes. Delays in report analysis directly increase debugging latency, impact release schedules, and reduce system reliability.

Although many automated systems exist for severity

prediction and duplicate detection, nature-based bug prediction remains relatively underexplored, particularly in multiclass settings where reports must be classified into distinct defect nature categories. The nature of a bug is crucial because it determines the debugging strategy, developer expertise requirement, and expected resolution workflow.

This research addresses that gap by proposing a Nature-Based Bug Report Prediction Model using ensemble machine learning techniques integrated with NLP preprocessing, TF-IDF vectorization, augmentation-driven robustness, and an XGBoost extension. The system is designed to automate nature prediction from textual bug descriptions and improve software maintenance efficiency.

The major contributions of this work are as follows:

- Comprehensive evaluation of baseline machine learning classifiers for nature-based classification
- ensemble voting framework for robust multiclass prediction
- text augmentation for class generalization improvement
- XGBoost extension for advanced predictive enhancement
- Benchmark validation on Eclipse and Mozilla repositories

II. RELATED WORK

Automated bug report analysis has been an active research area in software engineering, particularly in the domains of severity prediction, non-bug filtering, bug routing, and source-level localization

However, comparatively fewer studies address the problem of nature-based multiclass bug classification, which directly motivates the proposed research.

Polpinij proposed a method for non-bug report identification using one-class Support Vector Machine (SVM) [2]. The study compares multiple textual weighting schemes including TF, TF-IDF, and TF-IGM to separate valid bug reports from irrelevant or non-bug submissions. This work improves repository cleanliness and reduces manual triage effort. However, the classification task remains binary in nature and does not extend toward multiclass defect nature prediction.

In the area of bug routing and maintenance categorization, Adhikarla introduced an automated classification system for assigning reports into corrective and perfective maintenance categories [3]. The system employs keyword-driven textual features and classical classifiers, where SVM achieved high routing accuracy. While this significantly supports developer assignment and resource planning, the model does not classify the semantic nature of the defect itself, such as GUI, performance, configuration, or network related issues.

Source-level bug localization has also received considerable attention. Youm, Ahn, and Lee proposed the BLIA framework, which integrates textual bug descriptions, stack traces, structured file information, and code change histories for suspicious file ranking [4]. Their hybrid approach significantly improves mean average precision compared with existing localization systems. Despite its strong localization capability, the framework assumes that the bug nature is already understood and therefore does not solve the multiclass classification challenge targeted in this research.

Another supervised classification approach combines textual fields with additional structured attributes using TF-IDF, bigram extraction, feature selection, and KNN classification [5]. The study reports improved precision, recall, and F1-score for bug versus non-bug separation. However, the work remains focused on binary classification and does not address the richer multiclass problem of defect nature prediction.

The most closely related foundation to this research is the ensemble nature-based bug prediction model discussed in the base paper used for this project [6]. That work highlights the lack of robust nature-based multiclass classification systems and proposes ensemble learning with SVM, Random Forest, Logistic Regression, and voting mechanisms over Mozilla and Eclipse datasets. The present work extends this foundation further by incorporating text augmentation and an XGBoost enhancement layer, leading to stronger class generalization and improved predictive accuracy.

From the reviewed literature, it is evident that most prior studies focus on severity prediction [1], repository cleaning [2], maintenance routing [3], localization [4], and binary classification [5]. Very limited research specifically addresses nature-based multiclass bug prediction using ensemble

learning with boosting-based extensions [6]. This identified gap forms the core motivation of the proposed research.

III. SYSTEM ANALYSIS

The system analysis phase plays an important role in identifying the limitations of current bug report classification systems and defining the need for the proposed Nature-Based Bug Report Prediction Model. Modern software repositories such as Mozilla and Eclipse generate a large volume of bug reports containing textual descriptions, priority, severity, component details, and reproduction steps. As the number of reports increases, manual analysis becomes tedious, time-consuming, and prone to errors, which directly affects software maintenance efficiency.

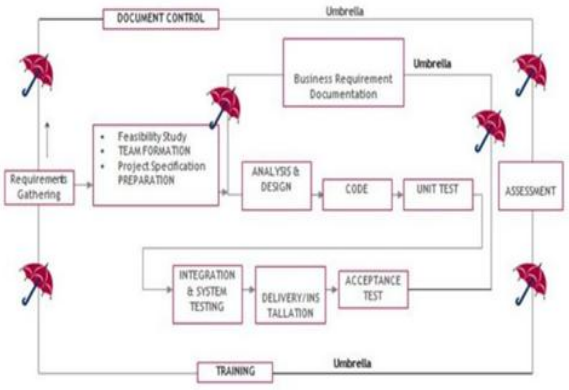

Fig. 1. SDLC Umbrella Model used for development lifecycle

The objective of this phase is to study existing bug classification approaches, identify their weaknesses in handling multiclass nature prediction, and justify the proposed ensemble-based solution.

A. Existing System

Existing systems mainly rely on traditional standalone machine learning algorithms such as KNN, SVM, Naive Bayes, and Random Trees for classifying bug reports. Some approaches focus on binary classification, where reports are categorized as bug or non-bug, while others classify them into corrective and perfective maintenance categories. A few studies also use Orthogonal Defect Classification (ODC) and limited multiclass categories such as concurrency, memory, and semantic bugs.

The major limitations of the existing system are:

- low robustness for large datasets
- weak multiclass classification performance
- absence of ensemble learning strategies
- no text augmentation support
- limited handling of overlapping bug categories
- no advanced boosting-based enhancement

These limitations reduce their suitability for real-world large-scale bug repositories.

B. Proposed System

The proposed system introduces a nature-based multiclass ensemble machine learning model that improves bug report prediction accuracy and stability. The workflow starts with uploading Eclipse and Mozilla bug datasets. The textual bug descriptions are preprocessed using special
symbol removal, stop-word elimination, stemming, and lemmatization to clean the data. The processed text is then converted into TF-IDF vectors, which numerically represent the importance of words in the reports.

The dataset is divided into 80% training and 20% testing sets. Multiple classifiers are then trained, including:

- Support Vector Machine
- Random Forest
- Logistic Regression
- Voting Ensemble Classifier
- XGBoost Extension

The Voting Ensemble Classifier combines predictions from multiple baseline models, improving multiclass stability and reducing the dependence on a single classifier. As an extension, XGBoost is added to further improve prediction performance, especially for overlapping and difficult bug categories.

The final system can accept unseen bug reports and automatically predict the corresponding bug nature through a user-friendly GUI.

C. Advantages

The proposed system offers several advantages over existing methods:

- improved multiclass classification

accuracy
- better robustness across datasets
- reduced manual bug triage effort
- support for six nature-based bug classes
- enhanced prediction through XGBoost
- scalable design for future deep learning integration

D. Feasibility Analysis

The proposed system is technically, operationally, and economically feasible. It is implemented using Python, Scikit-learn, NLTK, XGBoost, and Tkinter, which are open-source technologies. The system can run on standard hardware with 4 GB RAM and a basic processor, making it suitable for academic and real-world software maintenance use cases.

IV. SYSTEM DESIGN

The system design phase translates the analyzed requirements into a structured implementation framework for the proposed Nature-Based Bug Report Prediction Model. Since the project follows the SDLC Umbrella Model, the design process focuses on transforming functional requirements into modular components that support dataset processing, machine learning model execution, comparison, and prediction workflows.

The design of the proposed system is divided into four major stages: Designing Stage, Development Stage, Integration and Testing Stage, and Installation & Acceptance Stage. Each stage ensures that the system remains modular, maintainable, and scalable for future enhancement.

A. Designing Stage

The designing stage begins with transforming the identified requirements into a technical blueprint of the application. At this stage, the major modules of the system are finalized, including dataset upload, preprocessing, model training, comparison graph generation, and final bug nature prediction.

The user interface is designed as a Python Tkinter-based GUI, allowing users to interact with the system through simple buttons and workflow screens. The GUI includes modules for:

- uploading Mozilla and Eclipse bug datasets
- preprocessing textual bug reports
- executing SVM, Random Forest, Logistic Regression, Voting Classifier, and XGBoost
- generating comparison graphs
- predicting bug nature from unseen reports

The design also defines the data flow of the system, where raw textual bug descriptions pass through cleaning stages such as stop-word removal, stemming, and lemmatization before being converted into TF-IDF feature vectors.

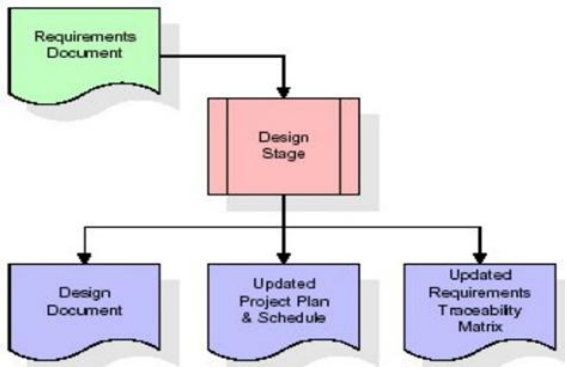

Fig 2: Designing Stage

This stage ensures that the system design is sufficiently detailed so that implementation can proceed with minimal ambiguity.

B. Development (Coding) Stage

In the development stage, the finalized design modules are converted into executable software components using Python.

The preprocessing engine is implemented using NLTK, Pandas, and Scikit-learn, where the uploaded dataset is cleaned and transformed into machine-readable TF-IDF vectors. Separate code modules are created for each machine learning classifier:
- SVM classifier module
- Random Forest module
- Logistic Regression module
- Voting Ensemble module
- XGBoost extension module

The GUI workflow is coded using Tkinter, where each button is linked to a dedicated backend function. For example, the dataset upload button invokes the preprocessing pipeline, while the comparison graph button triggers the Matplotlib visualization module.

This modular coding design improves maintainability

and allows future integration of deep learning or transformer-based NLP models.

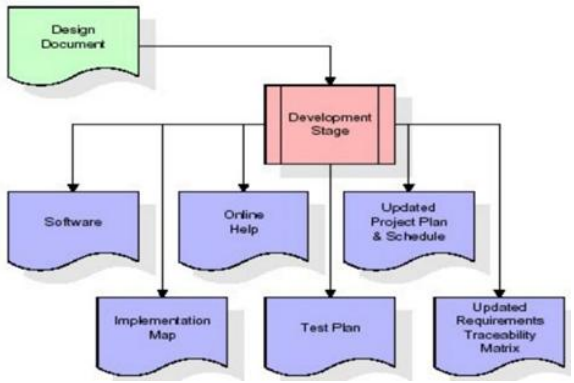

Fig 3: Development Stage

C. Integration and Test Stage

During the integration stage, all developed modules are combined into a unified workflow. The dataset upload module, preprocessing pipeline, machine learning classifiers, graph visualization system, and prediction interface are connected to ensure smooth data flow between components.

The integrated system is tested using Eclipse and Mozilla datasets. Each module is validated individually and then tested as part of the complete workflow. The major integration checks include:

- successful dataset upload
- correct preprocessing and TF-IDF conversion
- model training without runtime errors
- graph generation
- accurate bug nature prediction
- GUI response validation

This stage ensures that all modules work together as a single functional application.

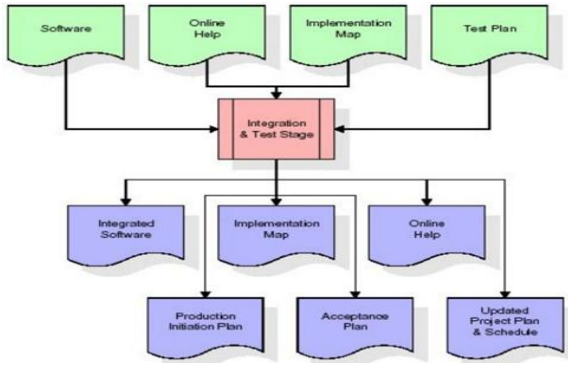

Fig 4: Integration and Test Stage

D. Installation and Acceptance Test

The final stage involves deploying the complete application into the target environment and validating it against expected outputs.

The system is installed on a standard Python-supported machine with required libraries such as:

- Pandas
- NumPy
- NLTK
- Scikit-learn
- XGBoost
- Matplotlib
- Tkinter

After installation, acceptance testing is performed by running the full workflow from dataset upload to final prediction. The results are compared with expected outputs to verify correctness.

This stage confirms that the system satisfies the project objectives of:

- automated nature-based bug classification
- multiclass prediction
- reduced manual triage
- improved maintenance efficiency

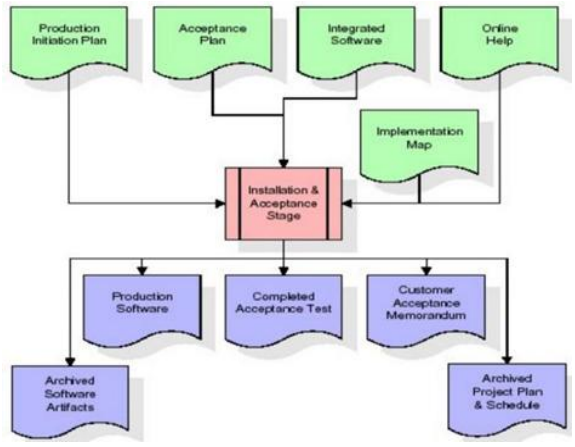

Fig 5: Installation & Acceptance Test

V. SYSTEM MODULES

The proposed Nature-Based Bug Report Prediction Model is divided into multiple functional modules to ensure modularity, maintainability, and accurate prediction of bug nature classes. Each module performs a specific task in the workflow, beginning from dataset upload and ending with final bug nature prediction. The modular design improves system

clarity and allows future enhancement without affecting the complete application.

A. Dataset Upload Module

The first module is responsible for uploading the Eclipse and Mozilla bug datasets into the application. The uploaded dataset contains textual bug descriptions and corresponding nature labels.

This module acts as the entry point of the system. Once the dataset is uploaded, the textual bug descriptions are extracted and passed to the preprocessing stage. The module also validates the dataset format and ensures that the required fields are present before processing begins.

B. Preprocessing Module

The preprocessing module is one of the most important stages of the system. Since bug reports are unstructured textual data, they must be converted into a clean and normalized format before training.

The preprocessing steps include:

- removal of special symbols
- tokenization
- stop-word removal
- stemming
- lemmatization
- text normalization

After cleaning, the text data is transformed into TF-IDF vectors, which numerically represent the importance of words in each bug report.

C. Support Vector Machine Module

This module trains the Support Vector Machine (SVM) classifier on the TF-IDF feature vectors.

SVM is used because it performs well on high-dimensional sparse text data and creates an optimal hyperplane for separating multiple bug nature classes. The trained model is then tested on unseen data to evaluate classification accuracy.

The SVM module serves as one of the major baseline classifiers in the system.

D. Random Forest Module

The Random Forest module trains an ensemble of decision trees using the processed training data.

This classifier improves robustness by combining multiple trees and performing majority voting for final prediction. It is especially useful for handling noisy textual features and multiclass classification problems.

E. Logistic Regression Module

This module implements Logistic Regression as a fast linear baseline classifier.

The model estimates the probability of each bug report belonging to a specific nature class and predicts the class with the highest probability. Logistic Regression provides a stable probabilistic baseline and is useful for comparing linear and non-linear model performance.

F. Voting Ensemble Module

The Voting Ensemble Module is the core prediction module of the proposed system.

This module combines the outputs of:

- SVM
- Random Forest
- Logistic Regression

The final prediction is generated using majority voting or weighted voting, which improves overall multiclass stability and reduces the dependency on a single classifier.

G. XGBoost Extension Module

As the extension contribution of the project, the system includes an XGBoost module.

XGBoost is a gradient boosting-based advanced machine learning algorithm that improves prediction performance by learning from previous classification errors. It provides stronger class boundaries and performs well on complex textual feature spaces.

H. Comparison Graph Module

The comparison graph module generates a visual performance comparison between all classifiers used in the system.

The module plots:

- SVM accuracy
- Random Forest accuracy
- Logistic Regression accuracy
- Voting Ensemble accuracy
- XGBoost accuracy

This graphical representation helps in analyzing the performance improvement achieved by the proposed ensemble and extension modules.

I. Bug Type Prediction Module

The final module allows users to upload unseen test bug reports and predict the corresponding bug nature. The input report passes through the same preprocessing and TF-IDF pipeline before being sent to the trained ensemble model. The predicted output is displayed through the GUI.

VI. IMPLEMENTATION

The proposed Nature-Based Bug Report Prediction Model is implemented entirely in Python 3.7 using open-source machine learning, NLP, visualization, and GUI libraries. The complete source code is modularized into dataset processing, classifier execution, ensemble integration, visualization, and GUI components, allowing independent testing, easier debugging, and future scalability.

A. Libraries Used

The implementation uses the following major Python libraries:

- Pandas: dataset loading, bug report manipulation, preprocessing, and structured data handling.
- NumPy: numerical computation, matrix operations, and TF-IDF vector transformations.
- NLTK: tokenization, stop-word removal, stemming, lemmatization, and text normalization.
- Scikit-learn: TF-IDF vectorization, train-test splitting, Support Vector Machine, Random Forest, Logistic Regression, Voting Classifier, and evaluation metrics.
- XGBoost: extension model for advanced boosting-based multiclass classification.
- Matplotlib: visualization of confusion matrices, comparison graphs, and

performance metrics.

- Tkinter: GUI for dataset upload, algorithm execution, graph generation, and bug prediction.
- Pickle / Joblib: storage and retrieval of trained model objects for future prediction.

This modular library selection ensures that the system remains lightweight and suitable for both academic and practical software maintenance environments.

B. XGBoost Extension Architecture

An XGBoost-based gradient boosting architecture is implemented as the major extension to the base ensemble learning model.

The extension model is trained on the same TF-IDF vectorized textual bug reports used by the baseline classifiers. The boosting architecture incrementally learns from previous classification errors and improves the separation of overlapping nature classes.

The XGBoost extension is particularly effective for:

- sparse high-dimensional TF-IDF vectors
- multiclass nature-based prediction
- minority class handling
- boosting difficult bug categories
- Reducing classifier bias

The extension is trained under the same dataset split conditions as the baseline classifiers to allow direct performance comparison.

C. GUI and Application Implementation

The client-side implementation is developed using a Tkinter-based graphical user interface,

which provides a simple workflow for executing the project.

The GUI includes the following primary functions:

1. Dataset Upload: users browse and upload the Eclipse Mozilla bug dataset.
2. Process Dataset: performs NLP preprocessing and TF-IDF conversion.
3. Run Algorithms: executes SVM, Random Forest, Logistic Regression, Voting Classifier, and XGBoost.
4. Comparison Graph: visualizes the performance of all classifiers.
5. Predict Bug Type: predicts bug nature for unseen test reports.

Each GUI button is linked to a dedicated backend Python function, ensuring clean separation between interface logic and classifier execution.

D. Data Preparation Details

The raw bug report dataset is collected from Mozilla and Eclipse repositories, containing multiple bug nature categories.

Each bug record contains:

- bug textual description
- class label
- repository source
- category type

The preprocessing stage applies:
- special symbol removal
- lowercase normalization
- stop-word elimination
- stemming
- lemmatization
- TF-IDF vector generation
- text augmentation

The final vectorized dataset is divided into 80% training and 20% testing data for model evaluation.
This preparation stage ensures that unstructured bug report text is transformed into a stable numerical representation suitable for multiclass machine learning classification.

VII. EXPERIMENTAL RESULTS

The proposed Nature-Based Bug Report Prediction Model was experimentally evaluated using the Mozilla and Eclipse bug report datasets. The experiments were designed to measure the performance of baseline classifiers, the proposed voting ensemble, and the XGBoost extension for multiclass nature-based bug prediction.
The evaluation was carried out using accuracy, precision, recall, F-score, confusion matrix analysis, and visual comparison graphs.

A. Dataset Overview

The dataset was collected from Mozilla and Eclipse bug repositories, containing multiple bug reports categorized into six nature-based classes:
- Program Anomaly
- GUI
- Network or Security
- Configuration
- Performance
- Test-Code

After preprocessing and text augmentation, the dataset was converted into TF-IDF feature vectors. The final vectorized dataset was divided into 80% training and 20% testing data for model evaluation.
The preprocessing stage ensured removal of noisy textual elements and improved lexical representation of bug descriptions.

B. Baseline Classifier Results

The first stage of experimentation evaluated standalone machine learning classifiers.
The Support Vector Machine (SVM) achieved 74% accuracy, demonstrating stable performance on sparse TF-IDF textual vectors.
The Random Forest classifier achieved 77% accuracy, outperforming SVM due to its tree-based ensemble learning mechanism and stronger handling of noisy feature interactions.
The Logistic Regression model achieved 70% accuracy, serving as the linear probabilistic baseline for comparison.
These results confirm that standalone classifiers provide acceptable performance, but their multiclass nature prediction capability remains limited.

C. Proposed Voting Ensemble Results

The proposed Voting Ensemble Classifier combines the outputs of SVM, Random Forest, and Logistic Regression.
Experimental results show that the ensemble achieved 89% accuracy, significantly outperforming all standalone models. The performance gain validates the strength of combining diverse classifiers for multiclass bug nature prediction.

D. XGBoost Extension Results

The XGBoost extension achieved the highest performance with 92% accuracy.
The boosting-based learning architecture improves:
- class boundary separation
- minority class learning
- overlapping category prediction
- bias reduction

This extension serves as the major enhancement over the base ensemble model and confirms the effectiveness of boosting for multiclass bug report classification.

E. Performance Comparison

The comparison graph presents the performance of:
- SVM
- Random Forest

- Logistic Regression
- Voting Classifier
- XGBoost

The graph clearly shows that the XGBoost extension achieved the highest accuracy, followed by the Voting Classifier.

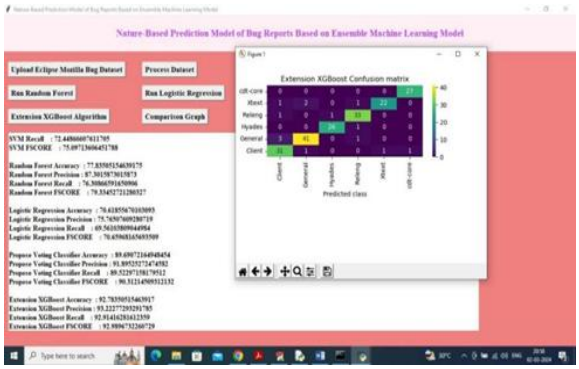

Fig 6: Comparison Graph

F. Final Bug Type Prediction Output

The final experiment evaluates the real-world usability of the proposed system.

Unseen test bug reports are uploaded into the system using the "Predict Bug Type from Test Data" function. The input text is processed through the same preprocessing and TF-IDF pipeline before classification.

The output screen displays:

- original bug report text
- predicted bug nature
- multiple sample predictions
- classification flow indicator

This confirms the practical applicability of the system for automated bug triage.

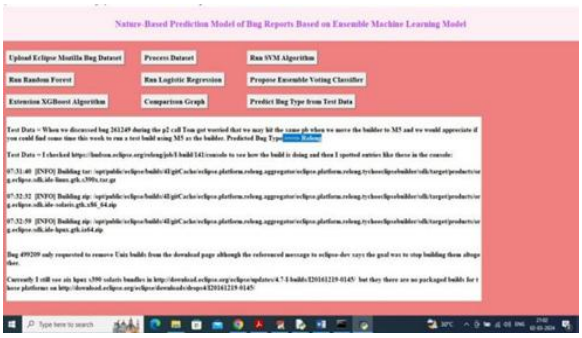

Fig 7: Final Prediction

G. Software Testing

The system was validated using three levels of testing:

- Module Testing: each classifier and preprocessing component tested independently
- Integration Testing: full workflow from dataset upload to prediction
- Acceptance Testing: final validation of the 89% ensemble and 92% XGBoost accuracy targets

All major modules successfully passed functional validation, confirming that the system satisfies the objective of automated nature-based bug classification.

VIII. CONCLUSION

This paper presented a Nature-Based Bug Report Prediction Model using ensemble machine learning techniques for automated multiclass classification of software bug reports. The system was developed to reduce the manual effort involved in analyzing large volumes of bug reports generated from Mozilla and Eclipse repositories.

The proposed workflow includes dataset preprocessing, TF-IDF vector generation, baseline classifier training, voting ensemble learning, and an XGBoost extension module. The preprocessing stage successfully transformed unstructured bug report text into machine-readable numerical vectors, enabling effective multiclass classification across six bug nature categories.

Experimental evaluation showed that the standalone classifiers achieved moderate performance, where Support Vector Machine obtained 74% accuracy, Random Forest achieved 77%, and Logistic Regression achieved 70%. The proposed Voting Ensemble Classifier significantly improved the prediction accuracy to 89%, proving the effectiveness of combining multiple machine learning models for robust bug nature classification.

As the advanced extension of the project, the XGBoost model achieved the highest accuracy of 92%, outperforming all baseline and ensemble models. This confirms that boosting-based learning is highly effective for handling complex and overlapping bug nature classes.

The final prediction module further demonstrated the real-world applicability of the system by successfully

classifying unseen bug reports through the GUI interface. This makes the proposed model useful for automated bug triage, faster software maintenance, and improved debugging workflows.

IX. FUTURE SCOPE

The future scope of this work includes:

- integration of deep learning models such as CNN, RNN, and LSTM
- transformer-based bug report understanding using BERT and CodeBERT
- real-time bug triage integration with issue tracking systems
- duplicate bug detection and severity prediction
- developer assignment recommendation
- multilingual bug report classification
- repository-aware intelligent debugging assistants

Overall, the proposed system demonstrates that ensemble learning combined with XGBoost can significantly improve nature-based bug report classification accuracy, making it a scalable solution for modern software maintenance environments.

ACKNOWLEDGEMENT

The authors express their sincere gratitude to Ms. Neha Shireen, Assistant Professor, Department of CSE-AIML, Lords Institute of Engineering and Technology, for her valuable guidance, continuous support, and encouragement throughout the development of this research work.

The authors also thank the Department of CSE-AIML, Lords Institute of Engineering and Technology, for providing the necessary facilities, resources, and academic environment required for the successful completion of this project.

Special appreciation is extended to all team members for their dedicated efforts, coordination, and contribution in the implementation and documentation of the proposed Nature-Based Bug Report Prediction Model.

REFERENCES

[1] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in Proc. 6th Int. Conf. Information and Communication Technology for the Muslim World (ICT4M), Nov. 2016, pp. 177-182

[2] Dr.Kamel Alikhan; Narasimhulu, C. V.; Reddy, K. Nagi; Fatima, Rayeez, 2022. Machine Learning Model Development based on Brazil's Covid-19 Data Set. | EBSCOhost.

[3] W. Wen, "Using natural language processing and machine learning techniques to characterize configuration bug reports: A study," M.S. thesis, College of Engineering, University of Kentucky, Lexington, KY, USA, 2017.

[4] J. Polpinij, "A method of non-bug report identification from bug report repository," Artificial Life and Robotics, vol. 26, no. 3, pp. 318-328, Aug. 2021.

[5] S. Adhikarla, "Automated bug classification: Bug report routing," M.S. thesis, Faculty of Arts and Sciences, Dept. of Computer and Information Science, Linköping University, Linköping, Sweden, 2020.

[6] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," Information and Software Technology, vol. 82, pp. 177-192, Feb. 2017.

[7] N. Safdari, H. Alrubaye, W. Aljedaani, B. B. Baez, A. DiStasi, and M. W. Mkaouer, "Learning to rank faulty source files for dependent bug reports," in Proc. SPIE, vol. 10989, 2019, Art. no. 109890B.

[8] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting," Sensors, vol. 19, no. 13, p. 2964, Jul. 2019.

[9] A. Aggarwal, "Types of bugs in software testing: 3 classifications with examples," May 2020.[Online].Available: https://www.scnsoft.com/software-testing/types-of-bugs

[10] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," Procedia Computer Science, vol. 132, pp. 352-361, Jan. 2018.

[11] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in Proc. 9th Int. Conf. Information and Communication Management, Aug. 2019, pp.

17-21.

[12] P. J. Morrison, R. Pandita, X. Xiao, R. Chillarege, and L. Williams, “Are vulnerabilities discovered and resolved like other defects?” Empirical Software Engineering, vol. 23, no. 3, pp. 1383-1421, Jun. 2018.

[13] F. Lopes, J. Agnelo, C. A. Teixeira, N. Laranjeiro, and J. Bernardino, “Automating orthogonal defect classification using machine learning algorithms,” Future Generation Computer Systems, vol. 102, pp. 932-947, Jan. 2020.

[14] T. Hirsch and B. Hofer, “Root cause prediction based on bug reports,” in Proc. IEEE Int. Symp. Software Reliability Engineering Workshops (ISSREW), Oct. 2020, pp. 171-176.

[15] Q. Umer, H. Liu, and I. Illahi, “CNN-based automatic prioritization of bug reports,” IEEE Transactions on Reliability, vol. 69, no. 4, pp. 1341-1354, Dec. 2020.

[16] H. Bani-Salameh, M. Sallam, and B. Al Shboul, “A deep-learning-based bug priority prediction using RNN-LSTM neural,” e-Informatica Software Engineering Journal, vol. 15, no. 1, pp. 1-17, 2021.

[17] Ö. Köksal and B. Tekinerdogan, “Automated classification of unstructured bilingual software bug reports: An industrial case study research,” Applied Sciences, vol. 12, no. 1, p. 338, Dec. 2021.

[18] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, “Learning to rank developers for bug report assignment,” Applied Soft Computing, vol. 95, Oct. 2020, Art. no. 106667.

[19] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, “Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts,” Empirical Software Engineering, vol. 21, no. 4, pp. 1533-1578, Aug. 2016.

[20] X. Ye, R. Bunescu, and C. Liu, “Learning to rank relevant files for bug reports using domain knowledge,” in Proc. 22nd ACM SIGSOFT Int. Symp. Foundations of Software Engineering, Nov. 2014, pp. 689-699.