Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume12, Issue 8, July 2021: 14458-14467

Predicting software failures during the development phase using machine learning

[1]A Suresh,[1]R G Kumar, [2]K.Harika, [2]D.Harshitha, [2]K.Harshitha, [2]T.Darshak

[1]Associate Professor, Dept of CSE, Siddharth Institute of Engineering & Technology, Puttur, Andhra Pradesh, 517583, India.
[2]UG Student, Dept of CSE, Siddharth Institute of Engineering & Technology, Puttur, Andhra Pradesh, 517583, India.
csesuresh6@gmail.com

Abstract—Software flaws were common through application development or could create a variety of issues for both consumers as well as engineers. As a result, different strategies were used by researchers to alleviate the effects of these faults in the program code. Among the most important methodologies concentrates upon fault prediction employing machine learning approaches, that could aid programmers in preventing problems from starting the construction ecosystem. Alternative methods for predicting the risk of flaws are presented in these publications. However, the majority of these studies attempt to predict faults from a large range of software attributes. Another major flaw in the recent literature has been the lack of a convincing explanation about whatever causes programs to become flawed. We employ the Extreme Gradient Boosting (XGBoost) methodology, which takes a training number of data for easy-to-compute component attributes as input but also determines if the relevant segment was defect-prone. Humans provided a brief pattern sampling strategy that selects accurate estimates with the smallest set of attributes to take advantage of the connection across prediction performance as well as concept interpretability. Our main premise would be elements that don't contribute to the designer's prediction accuracy shouldn't be shown. Surprisingly, the decreased number of characteristics aids design explainability, which is critical for providing developers more information on attributes associated with each component of code that would be more defect-prone. Humans test our methods on a variety of programs using statistics from leading companies, but they discover as (I) the characteristics that assist the most to identify the best forecasts vary depending on the problem, and (ii) it is feasible to create effective approaches with few characteristics that are easier to grasp.

Keywords— Software Development Life Cycle, XG Boost, Machine Learning, Bug Prediction, the Design phase

1. Introduction

Nowadays, completing a software project became a huge challenge [1]. Issues or glitches are a program owner's worst nightmare. These bugs happen as a consequence of bad code formulation and construction. The depth of understanding [2] has to be the most significant difficulty in writing defect-free code. Consider a team of 5-6 programmers working on a project, some of whom are seasoned or those who are newer [3]. The software employee now

has limited expertise with the types of errors that can occur in this code in real-life situations. As a result, they simply implement the project without regard for future issues. After the program was delivered to consumers, they would encounter defects in a non-uniform ecosystem, affecting the application's score, consumer experience, or effectiveness [4].

After that, a lot of effort and resources were spent to correct the errors. These problems or flaws could sometimes be considered as a weakness, as well as attackers would take advantage of this to attack the internet or program, compromising crucial information or income. As a result, it elucidates the difficulty that the computer industry faces, as well as the relevance of forecasting software faults throughout the development stage [5]. So that we would take the proper precautions before these impacts manifest themselves. The most difficult task in the computer industry would be to create a bug-free program. Even if software development businesses kept on monitoring, resolving this problem was challenging [6].Although any program created by a human wasn't an advanced method, defects are a common and important occurrence. Nonetheless, application development firms place a premium on early fault discovery via a variety of examinations as well as testing techniques. As a result, the issue was resolved; therefore they looked at a few other approaches based on machine learning [7].

2. LiteratureReview

He examined the majority of machine learning techniques, including both supervised and unsupervised methods, throughout this Methodology. The WEKA Tool was employed for the investigation, as well as the data set from the business was used to train the model [8]. For accurate determination, a recovery or categorization approach based on Convolution Neural Networks (CNN) as well as Long Short-Term Memory (LSTM) was developed [9]. Using historical information, proposed a method based on Supervised Learning methods such as logistic regression, Nave Bayes, and also Decision Tree (DT). In addition, the K-fold cross-validation technique [10] was applied. Outlier detection or removal was prioritized, followed by feature reduction [11].Presented fault identification as a binary classification problem, e.g., right or wrong, but used the deep Defects architecture to build a classifier that could identify faulty code over correct code. A flaw sensor system was proposed that works with several compilers or languages, including java, GCC, and Visual Studio. [12]. Using marginal R square scores, developed a scheme that uses the smallest and most correct quantity of executing measures at a period. The Eclipse JDT Core database [13] was chosen. Using One-Class Support Vector Machine (SVM), presented a one-class Software Fault Prediction (SFP) methodology [14]. Machine learning was used to forecast the susceptibility of a web service. This article generates input validation and sanitation properties. For each sink, it calculates a static backward slice.A web application's data flow graph, instruction dependence graph, as well as connection dependency graph [15], should be used to assess the software. The proposed methodology would be to prioritize the issues based on severity and element characteristics using the Clustering technique employing Bayes Net Classifier [16]. KC1, MC1, AR1, AC6, MC2 were used to train the model, which was compared to the findings of naive Bayes or j48 [17]. Purpose 10 Data Sets using Supervised Learning Bagging, SVM, DT, as well as Random Forest (RF) processors are among the tools used by the organization [19]. The information would be in the form of object-oriented matrices and

acquired from an open-source code. Genetically based Classification Systems [20] could be model introduced.

3. Objectives

- Web development companies could save money by catching flaws early.
- Therefore If customers access games that have problems, the gaming industry suffers significantly.
- Social media firms require such a system because they are the most susceptible to cyber assaults that could result in the loss of millions of users' data.

• Government entities must also have bug-free technology to keep confidential material out of the wrong hands.

4. Methodology

Throughout our investigation and extensive literature review, humans discovered that RF works well for Software Defects Forecast with a high degree of accuracy, but it could be further improved by another recently introduced algorithm-based XGBoost. In this study, we propose a strategy for constructing a statistical method with a higher degree of accuracy than RF.

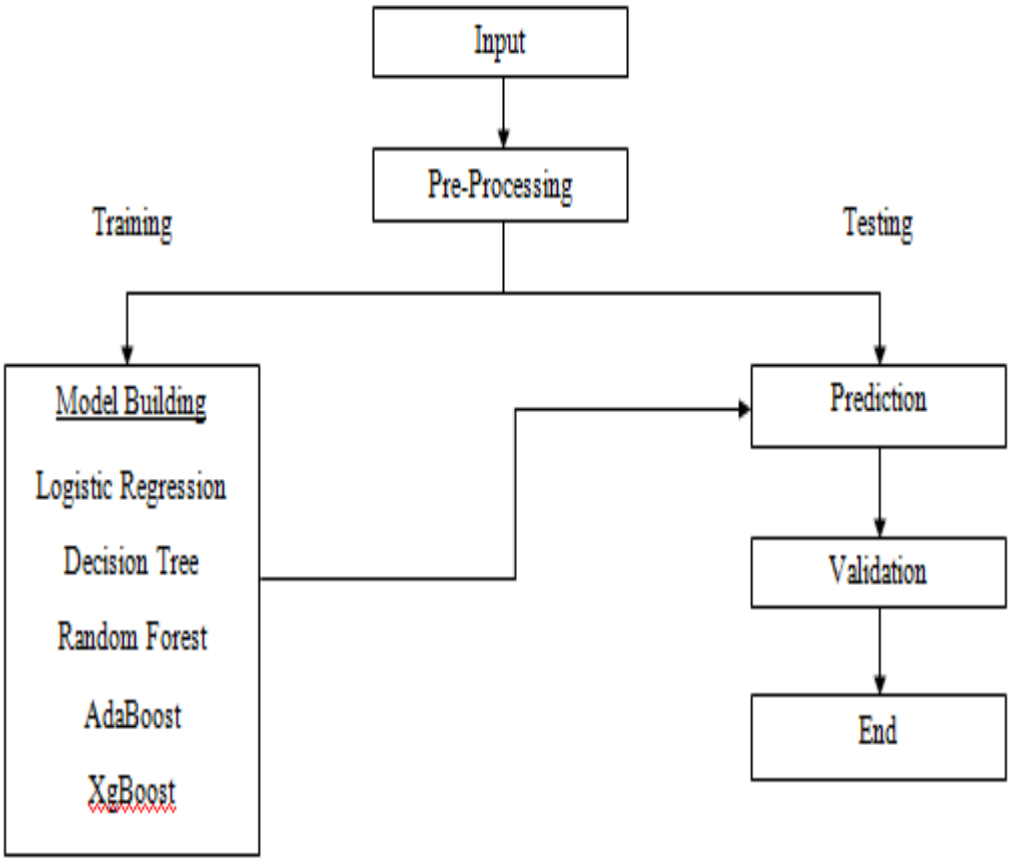

Figure 1: Workflow diagram

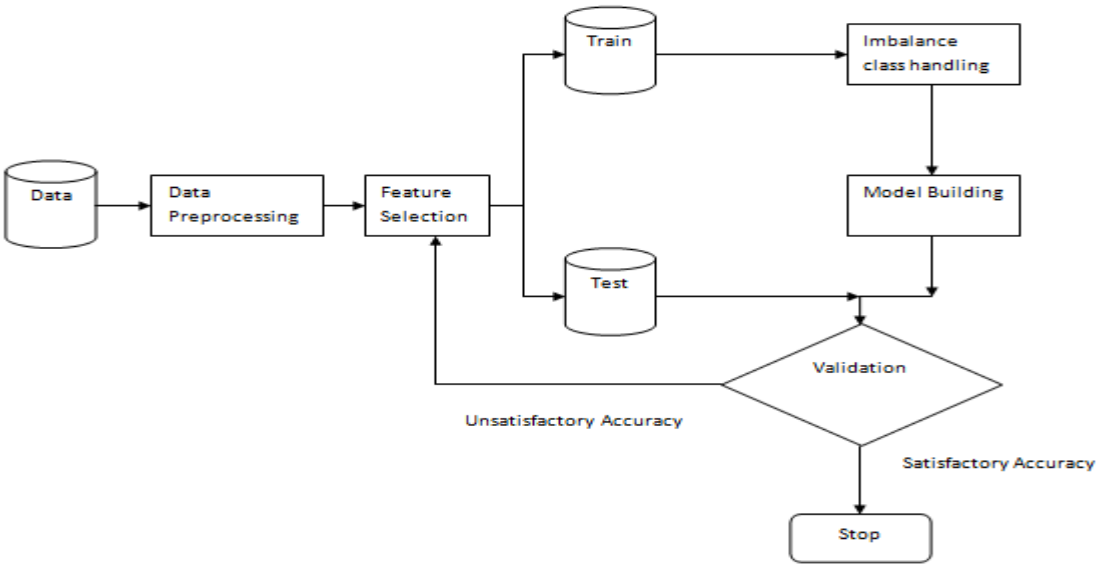

Figure 2: Workflow of the implementation stage

4.1 Data Prepossessing

The most interesting part of our period was machine learning. For their enterprises, everyone is starting to use machine learning algorithms. The information would be at the heart of the convoluted process. Our machine learning techniques ensure that the data we process would be of high quality. As a result, data pre-processing is an important stage in the construction of a good forecasting system.

The concept of features multiplication is often used to determine the distance of independent factors or data attributes. In our study, they used a normalization technique to bring our data to the same level.

Controlling Class Imbalance: As we would observe from our goal variable's scatter diagram. If our target variable is highly uneven, our forecasting model would neglect the minority class while being highly prejudiced toward the majority. In this research, they applied Synthetic Minority Over-sampling Technique (SMOTE), a synthetic minority oversampling method, to overcome this difficulty.

4.2 SoftwareDefectDataset

Humans chose a regular software problem database from the industry's Promise Repository to measure the accuracy of our technique. ABD, ML2, PR4, as well as OPD databases, are used. Except for ML2, which has 38 characteristics, each dataset comprises 22 characteristics. The following is a detailed explanation of the dataset:

The Metrics Information Program was used to create the business database. Sheppard et al. cleaned up the database in 2013 by removing duplicate as well as conflicting material. The database has this recommendation to improve. As a result, they used the cleansed information from the industry's Database in our research. For each event, the organization uses Halstead as well as McCabe statistics. Each project in the business has its range of attributes. Table 1 lists the particular amount of traits available.

Table 1: Datasets of the Organization

Dataset	Project Title	Instance	Number of Errors	Errors in %	Parameters
Company's	ABD	8965	1256	20.50%	26
	ML2	569	53	12.83%	24
	PR4	632	205	15.50%	24
	OPD	1254	265	12.80%	32

4.3 Data Split in Training Data & Testing Data

Humans can't use the dataset for training our algorithm. If they train our system overall data points, humans have a challenge with overfitting, therefore our algorithm may make an inaccurate forecast of a public statement. To assess the usefulness but also dependability of our strategy, humans agreed to separate our dataset into two sections with 80:20 ratios. The classifier is constructed using 80% of the data set, while the remaining 20% was utilized to assess the prediction performance using projected and real value connections.

5. Results and Discussions

To verify and compare the outcomes of other ways with the approach they provided, humans examined all of the models. Researchers tried five methods in this investigation: linear regression, DT, RF, ADA Boost, as well as Tuned XGBoost. Tuned XGBoost would be a proposed method in which N estimator, training process, max depth, and subsample characteristics were tuned across four datasets: ABD, ML2, PR4, and OPD.

A use sequence diagram would be a type of behavioral diagram specified and derived from a Use-case assessment in the Unified Modeling Language (UML). Its goal is to offer a visual representation of a system's functionality in terms of players, objectives (expressed as use instances), as well as any relationships between those, use instances. A use scenario diagram's principal aim would be to indicate which system activities are conducted for which player. The roles of the system's players are represented in Figure 3.

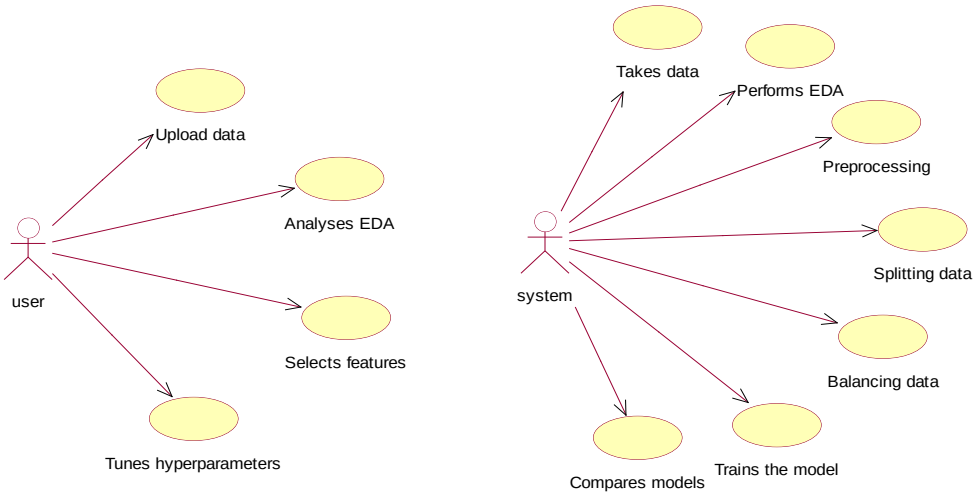

Figure 3: Proposed design case diagram.

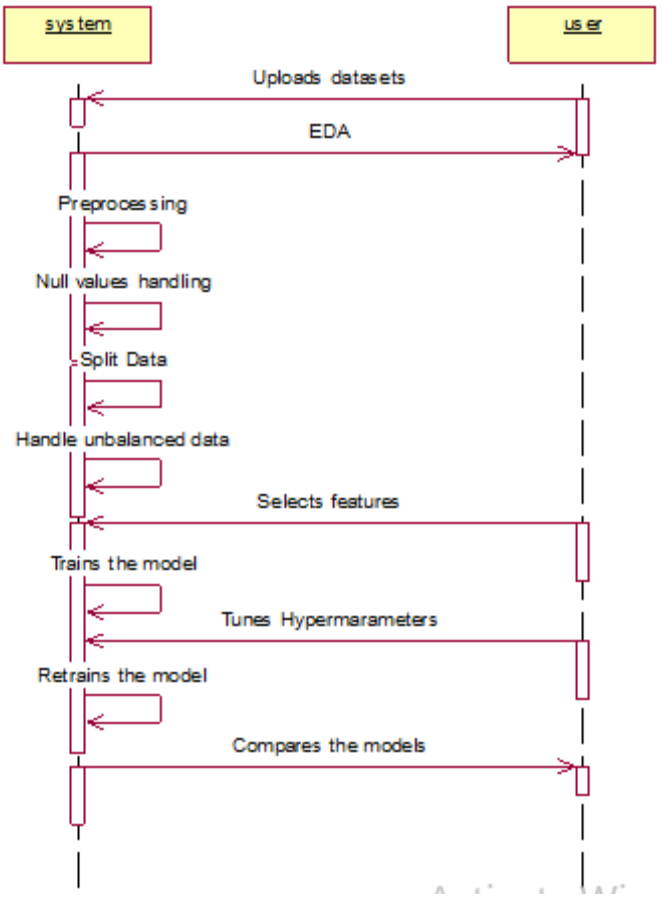

Figure 4 shows a diagram of the fault identification process.

In the Unified Modeling Language (UML), a sequence diagram is a type of interaction diagram that depicts how activities engage with one another and in what order. It's a Message Sequence Chart construct. Figure 4 shows a flow chart, also known as an event chart, an event situation, or a scheduling chart.

Researchers measured all models that included the goal characteristic to establish the forecasting ability of computer characteristics. The prediction accuracy in this instance is defined as the average AUC among all algorithms that included the goal attribute. Similarly, all models that use the program concept's average Mean Absolute Deviation (MAD) statistic account for volatility. Our application (USXGB) produced millions of models once more. Figure 3 depicts the objective software characteristics' predictive performance as well as variability. Particularly, roughly 3.5 percent of the characteristics were found in systems with an average AUC of more than 82 percent. Our method of feature selection resulted in the majority of the characteristics having much lower average AUC values (only around 77 percent).While examining the dissemination of characteristics while taking design variability into account, they notice a comparable tendency. Around 3% of the characteristics in this scenario are linked to systems with minimum disturbance. 73 percent of characteristics in the best-performing algorithms in terms of accuracy (AUC numbers) were related to Object-Oriented metrics, while the remaining 27% are related to CK metrics. Particular characteristics such as Measure Functional Abstraction (MFA) were found in roughly 14% of the best systems. Lines of Code (LOC) occurred in about 12 percent of the designs, while

Cohesion Among Methods of a Class (CAM) featured in nearly 11 percent of the designs, rounding out the top three.

In Figures 5–7, the findings obtained in state-of-the-art models have discussed our method.

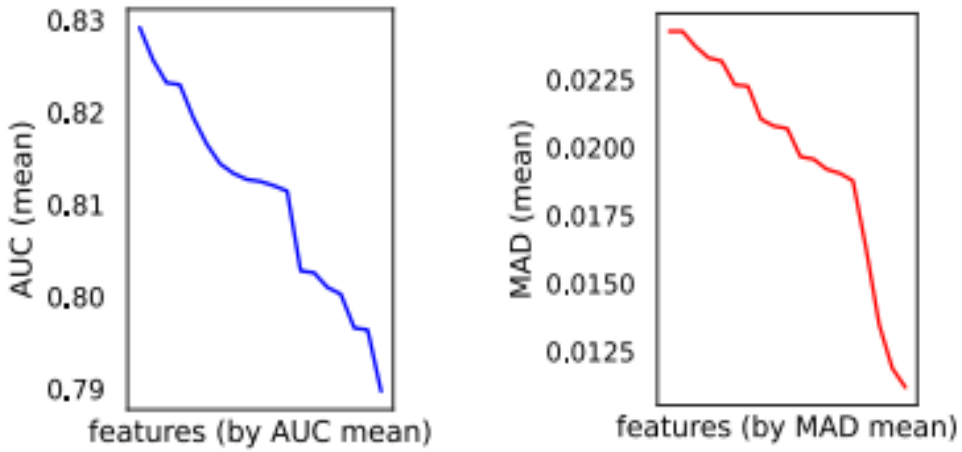

Figure 5: Prognosis A suggested approaches' reliability.

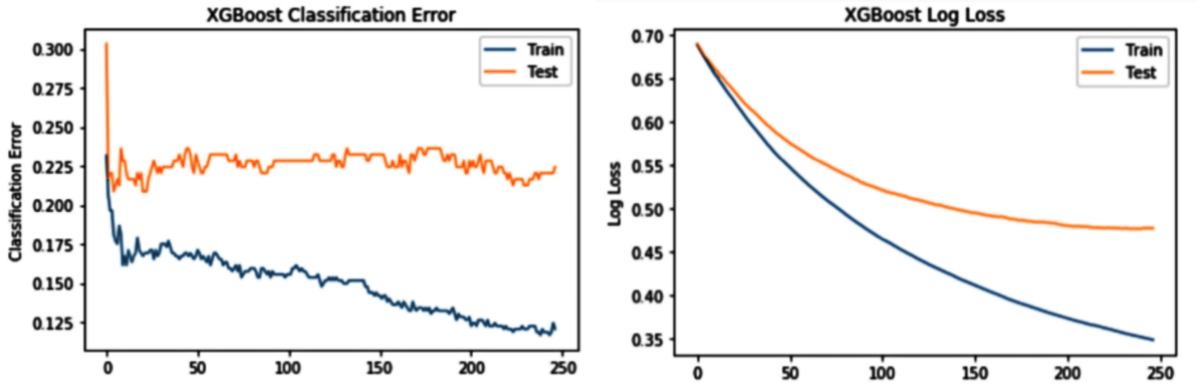

Figure 6: The database was tested and trained.

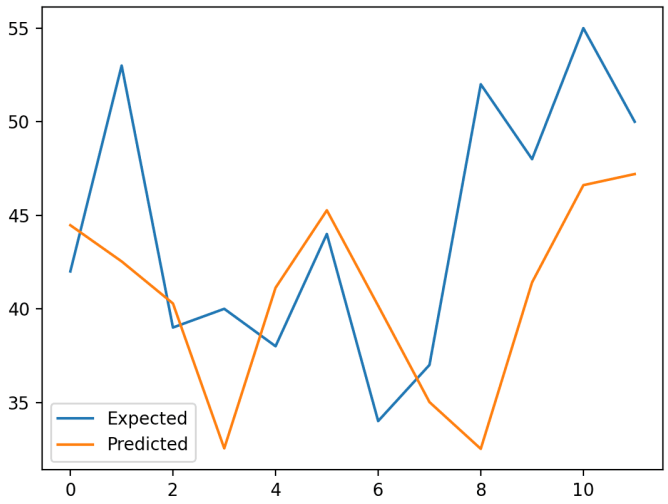

Figure 7 shows the XGBoost outcomes.

The trials with predicted precision, as well as variance, supported two key aspects of our

design selection method. First, depending on AUC figures, a limited set of criteria yields accurate estimates. Second, the program measures used by the most accurate models differ significantly. Object-Oriented characteristics were prevalent in the top-performing categories, accounting for over 68 percent of the key methods in terms of variation but nearly 75 percent in terms of effectiveness.

6 Reliability Issues As we'll see later, the research presented in this paper contains numerous flaws that could jeopardize our findings. Humans begin by describing the possible threats to authenticity. After that, they go into internal dangers. Furthermore, they look at the challenges to structure credibility or outcome accuracy.

6.1 External Validity: Factors that restrict our capacity to generalize the outcomes of our research were known as challenges to external validity. The small number of innovative they looked at in our analysis represents a danger to our research's external validity. In addition, all of the applications were based on the Java programming language. As a result, the findings may not apply to other applications, particularly those written in multiple programming languages. Moreover, our outcomes are influenced by flaws in the development's surroundings. As a result, they were unable to make any conclusions about cross-project faults.

6.2 Internal validity: Impacts that potentially affect the independence variable's causation are risks to internal consistency. This danger in our scenario pertains to the databases that blindly implemented the data provided. However, they were unable to verify the information in form of how the writers received it. For instance, the information could be inadequate or incorrectly gathered. Researchers use a machine learning approach like cross-validation to limit the consequences of unbalanced data, but we can't mean that the information represents the characteristics of the eight Java applications they used in our analysis.

6.3 Construct validity: Structural validity refers to the ability to apply the results of tests to a notion or idea. SHAP parameters are accepted as an impartial technique to describe machine learning algorithms in the present research. Other approaches in the literature, on either hand, could have alternate theories based on a set of features. For example, in the present literature on prediction models, LIME and BreakDown have already been studied.

7. Conclusions

Using an effective performance of the XGBoost method called USXGBoost, we searched the space for software estimation models, resulting in millions of random designs. The reliability or understandability of these algorithms was assessed. In the Jureczko databases, they discovered that 3.5 percent of the algorithms (out of 1 997 287) outperform the seven basic baseline models. Our findings also showed that software fault forecasting is a project-specific challenge, implying that the elements that make up the top-performing algorithms could differ significantly based on the assignment. As a result, it's crucial to comprehend the determinants that affect model judgments. As a result, concept descriptions may reveal which aspects of the code are more likely to fail.Humans plan to harvest information on public Github sources in the future. The same approaches used in this study might be used to identify but also evaluate this information. As a result, they would provide additional case

histories of the paradigm described in this work to the audience. A qualitative investigation with programmers to see how the descriptions offered in this article might benefit real projects would be another case study to verify the existing research. The findings of this research might be utilized to propose a tool for builders to examine their applications, but also determine which traits may suggest problematic classes.

References

[1] Kallis, R., Di Sorbo, A., Canfora, G., & Panichella, S. (2019, September). Ticket tagger: Machine learning-driven issue classification. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 406-409). IEEE.

[2] Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H. H., & Crnkovic, I. (2019, May). A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In International Conference on Agile Software Development (pp. 227-243). Springer, Cham.

[3] Pospieszny, P., Czarnacka-Chrobot, B., & Kobylinski, A. (2018). An effective approach for software project effort and duration estimation with machine learning algorithms. Journal of Systems and Software, 137, 184-196.

[4] Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., & De Lucia, A. (2018, March). Detecting code smells using machine learning techniques: are we there yet?. In 2018 IEEE 25th international conference on software analysis, evolution, and reengineering (saner) (pp. 612-621). IEEE.

[5] Banu, J. F., Muneeshwari, P., Raja, K., Suresh, S., Latchoumi, T. P., & Deepan, S. (2022, January). Ontology Based Image Retrieval by Utilizing Model Annotations and Content. In 2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 300-305). IEEE.

[6] Karnan, B., Kuppusamy, A., Latchoumi, T. P., Banerjee, A., Sinha, A., Biswas, A., & Subramanian, A. K. (2022). Multi-response Optimization of Turning Parameters for Cryogenically Treated and Tempered WC–Co Inserts. Journal of The Institution of Engineers (India): Series D, 1-12.

[7] Azeem, M. I., Palomba, F., Shi, L., & Wang, Q. (2019). Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology, 108, 115-138.

[8] Manjula, C., & Florence, L. (2019). A deep neural network-based hybrid approach for software defect prediction using software metrics. Cluster Computing, 22(4), 9847-9863.

[9] Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., & Wilson, J. (2019). The what-if tool: Interactive probing of machine learning models. IEEE transactions on visualization and computer graphics, 26(1), 56-65.

[10] Aroulanandam, V. V., Latchoumi, T. P., Bhavya, B., & Sultana, S. S. (2019). Object detection in convolution neural networks using iterative refinements. architecture, 15, 17.

[11] Torabi, M., Mosavi, A., Ozturk, P., Varkonyi-Koczy, A., & Istvan, V. (2018, September). A hybrid machine learning approach for daily prediction of solar radiation. In International Conference on Global Research and Education (pp. 266-274). Springer, Cham.

[12] Narmadha, R., Latchoumi, T. P., Jayanthiladevi, A., Yookesh, T. L., & Mary, S. P. (2022). A Fuzzy-Based Framework for an Agriculture Recommender System Using

Membership Function. In Applied Soft Computing: Techniques and Applications (pp. 207-223). CRC Press.
[13] Vanga, M. S. R., Vijayaraj, J., Kolluru, P., & Latchoumi, T. P. (2022). Semantics-Driven Safety Measures in Distributed Big Data Systems on IoT. In Advanced Computational Paradigms and Hybrid Intelligent Computing (pp. 251-259). Springer, Singapore.
[14] Braiek, H. B., & Khomh, F. (2020). On testing machine learning programs. Journal of Systems and Software, 164, 110542.
[15] Himanen, L., Jäger, M. O., Morooka, E. V., Canova, F. F., Ranawat, Y. S., Gao, D. Z., ... & Foster, A. S. (2020). describe a Library of descriptors for machine learning in materials science. Computer Physics Communications, 247, 106949.
[16] Li, Z., Jin, X. Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. It Software, 12(3), 161-175.
[17] Garikapati, P., Balamurugan, K., Latchoumi, T. P., & Malkapuram, R. (2021). A Cluster-Profile Comparative Study on Machining AlSi 7/63% of SiC Hybrid Composite Using Agglomerative Hierarchical Clustering and K-Means. Silicon, 13, 961-972.
[18] Aroulanandam, V. V., Latchoumi, T. P., Balamurugan, K., & Yookesh, T. L. (2020). Improving the Energy Efficiency in Mobile Ad-Hoc Network Using Learning-Based Routing. Rev. d'Intelligence Artif., 34(3), 337-343.
[19] Garikipati, P., & Balamurugan, K. (2021). Abrasive Water Jet Machining Studies on AlSi 7+ 63% SiC Hybrid Composite. In Advances in Industrial Automation and Smart Manufacturing (pp. 743-751). Springer, Singapore.
[20] Balamurugan, K., Uthayakumar, M., Sankar, S., Hareesh, U. S., & Warrier, K. G. K. (2017). Mathematical modeling on multiple variables in machining LaPO4/Y2O3 composite by abrasive waterjet. International Journal of Machining and Machinability of Materials, 19(5), 426-439.