Journal of Universal Computer Science, vol. 31, no. 5 (2025), 469-493
submitted: 17/6/2024, accepted: 7/12/2024, appeared: 28/4/2025 CC BY 4.0

Identification of Fault Prone Components in Multimedia Software based on Optimal Threshold Values decided using Genetic Algorithm

Manpreet Singh
(National Institute of Technology, Kurukshetra, Haryana, India
https://orcid.org/0000-0001-7003-102X, manibhangu92@gmail.com)

Jitender Kumar Chhabra
(National Institute of Technology, Kurukshetra, Haryana, India
https://orcid.org/0000-0002-2257-0982, jitenderchhabra@gmail.com)

Abstract: Fault prediction of multimedia software is necessary to develop good quality multimedia software because integrating various multimedia heterogeneous components in a software system usually generates many faults. So, this research article proposes a new fault prediction model based on the decided threshold values of structural features. These features are captured using metrics specifically identified for multimedia software and weighted suitably based on the behavior of the components dealing with multimedia handling. The threshold values are optimized using the genetic algorithm (GA). This paper also proposes a GA-based technique to combine multiple features using conjunction (AND) and disjunction (OR) operators while finding threshold values. Finally, the proposed model is tested for cross-project software fault prediction on selected six multimedia software and validated on three other general software datasets. Results show that our identified thresholds-based model performs excellently for multimedia software and satisfactorily over other general software.

Keywords: Code Metrics, Fault Prediction, Genetic Algorithm, Threshold Values, Software Engineering, Optimization
Categories: D.2.5
DOI: 10.3897/jucs.129859

1 Introduction

Today's world is impossible without integrating multimedia services into software, as all digital devices support multimedia services such as live broadcasting, images, audio, and video playing [Rathee, 19]. The complexity of software with integrated multimedia services is also increasing due to the heterogeneous nature of the components, and hence, developing fault-free multimedia software is a challenging task for the development team with budget and time constraints [Pachouly, 22] [Kanwar, 23]. Therefore, researchers are attracted to the automation of the testing of multimedia software to make the software development process fast by reducing the testing team's effort, as the testing of the software consumes half of the development resources [Singh, 24]. Hence, many fault prediction models have been proposed in the literature [Liu, 23] [Rathee, 22] [Sharma, 23] [Tang, 24].

Most of the multimedia software fault prediction (MSFP) models are based on machine learning (ML), statistics, and threshold values of the code metrics extracted by analyzing the code of the multimedia software [Shatnawi, 10a][Shatnawi, 10b] [Gyimóthy, 05] [Malhotra, 15] [Singh, 14] [Prajapati, 22] out of which threshold-based are fastest, easily implementable and understandable by development team [Dejaeger, 12][Catal, 09][Bishnu, 11]. Threshold-based models decide a particular threshold for each code metric and consider the class faulty if the value of any metric exceeds the predefined threshold value of the metric. So, threshold-based SFP models give information on the faulty class and the reason for the fault by considering which metric threshold value is exceeded [Boucher, 18]. In the case of general software, coupling and cohesion are interdependent, which means cohesion of the software components improves with a decrease in coupling and vice versa. Conversely, the concept of high cohesion and low coupling is not true for the software with integrated multimedia services because the component of the software handling multimedia services such as audio, video, or image may be highly cohesive due to handling the same type of service and at the same time highly coupled with other components of the software. Similarly, the complexity of the component handling multimedia service depends upon the type of the service, such as component handling video services being more complex than components handling images. Subjective thresholds of the existing metrics are available and decided by experts such as McCabe, who decided the fixed threshold value for cyclomatic complexity, and Rosenberg, who decided the threshold values of CK metrics [Catal, 09][McCabe, 76][Rosenberg, 98]. However, Subjective threshold values can't be generalized for all types of projects because they may change based on the design of the software, the type of the software under development (multimedia software or non-multimedia software), the chosen language to write the code of the software, scale, and complexity of the software [Shatnawi, 10a]. Furthermore, research suggests that metrics threshold values should be different for different versions of the same software, especially for multimedia software [Malhotra, 15][Zhou, 06], and hence, there is a need to dynamically decide the threshold values for each version of multimedia software.

This paper proposes a new way to dynamically decide the optimal thresholds of structural code metrics and group the metrics for fault prediction of software with integrated multimedia services. We collect our proposed structural code metrics information for the classes of existing multimedia projects, normalize the values to standardize the scale, label the data based on the pre-trained model, and then the genetic algorithm is used to dynamically decide the simultaneous threshold values of the selected code metrics. Two operations, conjunction (AND) and disjunction (OR), are used to group the threshold values of the code metrics, and these operations are also optimized through the genetic algorithm. The proposed threshold-based MSFP model is tested for cross-project SFP by training on different multimedia software datasets and testing on different multimedia software datasets to show the superiority of the proposed model over existing code metrics-based models.

The rest of the paper is organized into four sections. Section 2 explains the previous work done by the researchers in multimedia software fault prediction. Section 3 explains the overall methodology of the proposed work to decide the collective thresholds. Section 4 explains the experiments done on the selected multimedia software with a detailed discussion of the experiment's outcome. Finally, a conclusion of the overall work and direction for future work is given, followed by references.

2 Related Works

Metrics thresholds identify the software fault by checking if any metrics' threshold value is violated [Boucher, 18]. So, metrics threshold-based fault identification models are simple and easy to implement. Further, these models also provide the reason for the fault by considering which metric's threshold value is violated [Malhotra, 15] as the aim of this study is to decide objective-based collective threshold values. So, This section presents the previous work for SFP based on the metrics' thresholds.

Initially, metrics threshold values were subjective and were decided by the expertes [Catal, 09b] [McCabe, 76] such as McCabe has suggested the subjective threshold value for his cyclomatic complexity [McCabe, 76], and Rosenberg has suggested threshold values for CK (Chidamber and Kemerer) metrics [Rosenberg, 98]. However, these subjective threshold values can't be generalized for all projects. They can vary from project to project based on the different coding styles followed by the development teams, different programming languages used by the developers, different coding standards used in different companies, and the scale of the project such as large, medium, or small. In fact, some studies suggest that metrics threshold values should be decided separately for each version of the same project too [Shatnawi, 10a] [Zhou, 06]. Hence, there is a need to calculate the thresholds based on objective values.

So, Bender et al. [Bender, 99] proposed the first objective-based threshold detection technique called VARL (Value of an Acceptable Risk Level). Although the original technique was not proposed for SFP [Boucher, 18] This technique was later adopted to find the threshold values for SFP [Shatnawi, 10b] [Malhotra, 15] [Singh, 2014]. This technique utilizes the concept of logistic regression (LR) on a single code metric to identify the threshold where the LR coefficient represents the optimal threshold value of the code metric. Arar et al. [Arar, 16] have done a study to replicate the results of VARL in the field of SFP. This study prioritizes the software components based on the risk of fault occurrence so that tests can focus on critical software components. The proposed technique works equally well on multimedia and general software.

Unlike VARL, Alves et al. [Alves, 10] developed the first technique called Alves ranking which was originally designed to decide the threshold values of software code metrics. This technique was designed to measure the quality of the code. However, it can be used for SFP because there is a direct relationship between the code quality and the probability of the fault. This is a six-step process to find the threshold values of the code metrics and rank them based on the effectiveness in fault prediction of multimedia and non-multimedia software. The proposed technique is very effective in the field of software engineering.

Shatnawi et al. [Shatnawi, 10a] developed a new receiver operating characteristic curve-based technique in 2010 to identify thresholds of the code metrics which is better than both VARL and Alves ranking. It draws a receiver operating characteristic curve for each code metric and finds the metric's value where the area under the curve is maximum, which is decided as the threshold value of that metric. Catal et al. [Catal, 11] have utilized the ROC method to propose a new noise removal method for SFP datasets (multimedia or general) based on metrics thresholds. Results collected from the NASA datasets proved the success of the proposed method.

Hence, the ROC curve, Alves ranking, and VARL are the most effective known methods to detect the structural features' threshold values. All three methods can also be adapted to find the thresholds for multimedia software. So, Boucher et al. [Boucher, 18] have compared these three threshold detection techniques in their comparative study. All three techniques are well-suited for multimedia software fault prediction. They compared the metrics threshold values-based fault prediction with machine learning-based (ML-based) SFP on selected twelve software datasets downloaded from the Promise repository for fault severity prediction. Results show that metrics threshold values-based models are faster than ML models and, in some cases, outperform ML models.

However, all these methods find independent thresholds of the code metrics, which leads to the degradation of the model's performance, which necessitates developing a paradigm to find the collective thresholds. So, Singh and Chhabra [Singh, 23] proposed a new threshold identification technique based on optimization algorithms that can be applied to general and multimedia software. Unlike ROC, VARL, and Alves ranking, this technique finds the collective threshold values of the code metrics which are better than independent threshold values. Particle swarm optimization and genetic algorithm are adapted for this purpose, and results show that the genetic algorithm finds better threshold values with less iteration.

Table 1 summarizes the three famous threshold detection techniques used in the literature for SFP. Table 1 also summarizes the strengths and weaknesses of these techniques, and other latest studies done based on these techniques.

Author	Methodology and strength	Weakness
Bender et al. [Bender, 99]	Bender et al. have proposed the first technique called VARL (Value under the Acceptable Risk Level) to find the threshold value of the code metrics. This technique was not originally developed for SFP but was adopted later for this purpose. This technique utilizes univariate logistic regression to find the threshold values of the code metrics. After fitting the regression line coefficient is used as the optimal threshold value of the selected code metric.	This technique was not originally developed to find the threshold value of software code metrics for SFP. However, many authors have adopted it for SFP. So, this technique doesn't provide the optimal threshold values of the code metrics to identify the faults.
Alves et al. [Alves, 10]	Alves et al. have proposed the first technique called Alves ranking, which was originally developed to measure the quality of the code based on decided threshold values. There is a direct relationship between the quality of the code and the probability of	Although this technique is good for identifying the metrics threshold values. It considers each code metric independent while calculating threshold value. But in a real scenario dataset consists of

	a fault in the code. So, it is very useful to identify the software faults along with the reason for the fault. This technique follows six steps to find the optimal threshold values of the code metrics.	multiple code metrics to capture all aspects of the code such as complexity, coupling, and cohesion. So, all the code metrics are interdependent and there is a need to find the collective threshold values.
Shatnawi et al. [Shatnawi, 10a]	Shatnawi et al. have proposed a technique based on ROC curve to drive the threshold values of the code metrics. In this technique an ROC curve is drawn for individual code metrics based on multiple threshold values of that code metric and the area under the curve (AUC) is calculated for each threshold value. Finally, the threshold with maximum AUC is selected as the optimal threshold value of the code metric. This technique is better than both Alves ranking and VARL [Boucher, 18].	This technique is faster than the Alves ranking and gives better threshold values than both the VARL and Alves ranking. However, it also considers all code metrics independent like Alves ranking and VARL while calculating the threshold values. So, there is a need to develop an algorithm to find the collective threshold values.
Arar et al. [Arar, 16]	Arar et al. have done a replicated study of VARL for SFP. In their study, they have performed two experiments. The first experiment is to decide the single threshold value for each code metric to identify the software fault. The second experiment is performed to find the multiple threshold values to identify the severity of the fault.	This study proves the practical significance of the VARL in finding the software defects-based metrics threshold values. However, this study is not done for cross-project and scenario changes in the case of cross-project software fault prediction because threshold values decided on one project may not be useful for other software.
Boucher et al. [Boucher, 18]	Boucher et al. have done a comparative study of VARL, ROC, and Alves ranking for software fault and its severity prediction. In the case of fault identification, only one threshold value per code metric is decided. In the case of severity prediction,	This study proves that ROC is the best technique from the literature to identify the faulty modules and the severity of the fault in the software. However, this study is not done for cross-projects. Further, all

	multiple thresholds per code metric are decided. They proved that the ROC-based threshold detection technique is most effective for SFP.	three selected techniques consider code metrics independent to find the threshold values, which is not desirable to find the optimal solution.
Singh and Chhabra [Singh, 23]	Singh and Chhabra proposed a metaheuristic-based threshold detection technique for SFP. Unlike VARL, ROC, and Alves ranking, this technique finds the collective threshold values by optimizing the sum of the true positive rate and the true negative rate, which is more practical. However, the experiment is not conducted for cross-project to find the validity.	The proposed technique finds better threshold values compared to ROC, VARL, and Alves ranking and considers all aspects interdependent while calculating threshold values. However, the technique is not designed for cross-project SFP.

Table 1: Summary of the existing studies for metrics threshold detection

Beyond these primary studies summarized in Table 1, there are some other SFP studies done by utilizing the code metrics threshold values which are explained below:

Yu and Mishra [Yu, 12] have done an experimental study to show the practical use of different complexity metrics in the field of SFP. They proved that the accuracy of the SFP model increases with an increase in dataset size but recall degrades. However, recall can be increased by lowering the threshold value but it increases false positives leading to an increase in maintenance cost. They have utilized binary logistic regression for this purpose. They also proved that the threshold values are project-dependent.

Sharma, D., & Chandra, P [Sharma, 24] have explored the usage of factor analysis with regression (FAWR) in the field of SFP and the quality of the technique is tested through two experiments. The results of the experiments proved that FAWR is superior to applying two techniques individually in terms of precision and recall.

Mishra et al. [Mishra, 21] have done a systematic mapping of the 45 existing studies to calculate the threshold values of the code metrics. They have identified that most of the existing studies are based on empirical analysis, and CK metrics have been used to identify the threshold values. They further identified that a considerable number of studies have not been validated in terms of quality metrics.

Colakoglu et al. [Colakoglu, 21] have done a systematic mapping of 70 research articles to produce quality metrics. A trend map is shown to show the growth of this field and usage of the emerging technologies such as machine learning, artificial intelligence, and the Internet of Things in this field.

Tang et al. [Tang, 24] proposed a new technique, LTHFFA (learnable three-line hybrid feature fusion), to mitigate the curse of dimensionality in SFP because many code metrics are available. In this method, three different feature selection methods are used to find the subset of code metrics; then, based on three-line hybrid breeding, selected feature sets are merged to find the optimal feature set. The proposed feature

selection technique results are compared with nine other feature selection techniques to show the superiority of the proposed feature selection method. This feature selection technique is also applicable for multimedia software because fault prediction in multimedia software is also carried out with similar metrics but with different scales (weights).

Liu et al. [Liu, 23] proposed a new semantic feature extraction technique based on natural language processing using transformer architecture. Extracted semantic features are combined with structural features and code history-based features to improve the fault prediction results for multimedia and general software. Results show that the proposed features-based model significantly improves the prediction result. Semantic features are not well suited for MSFP; hence, semantic features are not focused.

After reviewing existing literature, it is clear that the metrics' threshold-based models are faster than ML and quite easy to implement and understand. However, there are some drawbacks to these models.

1. It is necessary to identify the thresholds of code metrics for each project because threshold values are project-dependent.
2. Threshold values depend upon the complexity, size, type, scale of software and the chosen programming language for software development, as threshold values of multimedia software differ from general software for the same metrics due to the design structure of the multimedia software. Here, scale is a unified term used to represent the project size (number of classes), average class size (number of methods and their size), average coupling, and complexity of the software.
3. Based on the historical information, the combined threshold values of the metric suite and connectors (AND or OR) are needed. For example, cohesion and coupling are interdependent for general software and independent for multimedia software. So, identifying suitable connectors is necessary for better fault prediction.
4. The architecture of multimedia software is different from that of general software because human interaction involves GUI and event handling, and hence, all the aspects, such as complexity, coupling, cohesion, inheritance, and polymorphism, contribute differently to multimedia software. So, there is a need for the weighted contribution of the code metrics to cover different design aspects for better software fault prediction, such as cohesion, coupling, and complexity, which contributes more than inheritance and polymorphism for multimedia software.
5. There is a need to bridge the gap between multimedia and general software fault prediction using dynamic threshold values of the code metrics.

This research article proposes a new SFP model for multimedia software by deciding metrics threshold values dynamically using the genetic optimizer to mitigate all these challenges. The connectors (AND or OR) among the threshold values are also decided to be them simultaneously for fault prediction of multimedia software under development. The proposed software fault prediction model is explained in detail in the methodology section.

3 Methodology

This section presents the proposed threshold detection technique, where the genetic algorithm is used to identify the optimal threshold values of code metrics. Figure 1 presents the overall structure of the proposed model using a data flow diagram. This technique is designed for fault prediction in multimedia software using our structural code metrics TCC, CAC, LCOCC, IMCR, and OMCR to cover all object-oriented structural features of the software [Singh, 24a] [Singh, 24b]. These code metrics are used to develop threshold-based cross-project software fault prediction (CPSFP) model, because these metrics more effectively capture the structural features of the software compared to existing code metrics [Singh, 24a] [Singh, 24b] such as TCC uses weighted path complexity (WPC), an advanced form of cyclomatic complexity, to find the overall complexity of the class, CAC considers parameters passed in function calls along with their data types unlike existing coupling metrics which counts only incoming and outgoing edges, LCOCC utilize both private methods and data types of the private data memes while calculating the cohesion of the class, and IMCR & OMCR also utilizes WPC for the calculation of weightage of inherited and polymorphic methods per class instead of trivial counting of these methods. In the first step, it extracts the discussed structural code metrics information of all classes of the selected multimedia software and labels them as faulty or non-faulty based on the available information of general software by finding the most similar k classes where Euclidean distance is used to find the similarity between two classes. In the second step, it assigns the weights to all five code metrics to adjust the contribution of the metrics in fault prediction of multimedia software. In the case of general software, cohesion, and coupling are interdependent and inversely proportional. It means with an increase in coupling, the cohesion of the component decreases and vice versa. However, multimedia software has no such observations, and software components with high cohesion may also have high coupling and complexity. However, all three aspects, complexity, cohesion, and coupling, are more important than other object-oriented aspects, such as inheritance and polymorphism for multimedia software. Therefore, more weightage is given to the TCC, CAC, and LCOCC and less to IMCR and OMCR, which is discussed later in detail. The third step calculates the thresholds of the metrics and their connectors (AND or OR) based on the genetic algorithm for fault prediction. Finally, identified thresholds are used for the SFP of the target software (software under development).

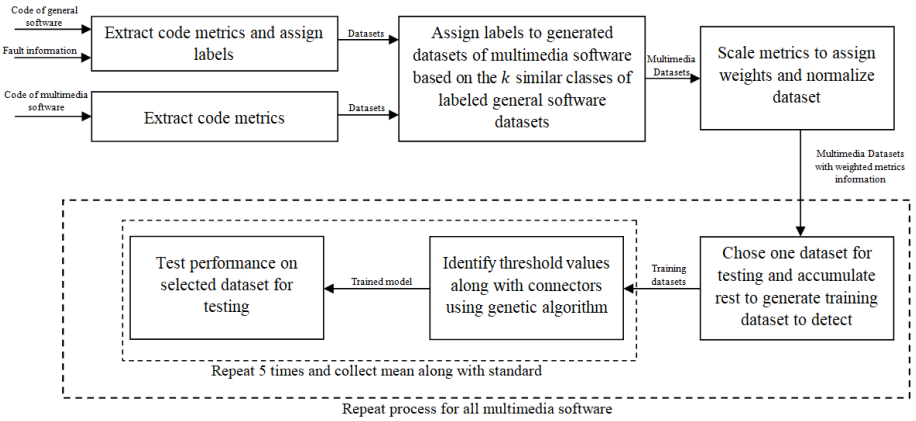

Figure 1: Block diagram of the proposed model for fault prediction in multimedia software

3.1 Preparing the Labelled Dataset of Multimedia Software

This is the first phase of the proposed fault prediction model, where our code metrics-based datasets are prepared of the selected multimedia software shown in Table 2, and our five metrics used to prepare datasets are shown in Table 4 of the result section.

The first step of this phase extracts our code metrics' information for all classes of the selected multimedia software to generate unlabelled datasets $X \in \mathbb{R}^{n*5}$ for all selected multimedia software, which consists of n rows and five columns where each column represents some code metric (attribute of the dataset) and each row represents our code metrics' information of some class. The second step assigns the labels $y \in \{0,1\}^n$ to all rows of the selected multimedia software datasets by finding the most similar k classes from the existing labelled software datasets. The similarity of the classes is calculated based on Euclidean distance shown in Equation (1), and the value of k is decided by applying 5-fold cross-validation on the existing labelled datasets.

$$D\left(x^{M(i)}, x^{E(j)}\right) = \sqrt[2]{\sum_{k=1}^{5}\left(x_k^{M(i)} - x_k^{E(j)}\right)^2} \quad (1)$$

In equation (1), $D\left(x^{M(i)}, x^{E(j)}\right)$ represents the Euclidean distance between i^{th} instance $x^{M(i)}$ of multimedia software dataset and j^{th} instance $x^{E(j)}$ of existing labelled dataset.

After finding the Euclidean distance from all the instances of the existing labelled dataset, top k instances are selected, and the label is determined using majority voting to the multimedia software's dataset instance. This process divides the multimedia software datasets into two clusters where one cluster represents the faulty instances (classes), and the other represents the non-faulty instances, excluding outliers. The third step scales the code metrics values based on equation (2).

$$x_i \in X^M = \{0.3 * TCC, 0.3 * CAC, 0.3 * LCOCC, 0.05 * IMCR, 0.05 * OMCR\} \forall x_i, 1 \leq i \leq n, \quad (2)$$

In equation (2), $x_i \in X^M$ represents the i^{th} instance of the multimedia software's dataset generated based on code metrics shown in Table 4. While scaling the dataset, TCC, CAC, and LCOCC have given more weightage than IMCR and OMCR. Most of the classes in the multimedia software have no inheritance or are derived from in-built classes that are stable and have no fault. Further, there is minimal method and operator overloading usage in the multimedia software's code because most methods are designed to handle independent events, leading to the event-specific name of each method. So, inheritance and polymorphism play little role in fault prediction for multimedia software. The statistics of the percentage classes involved in inheritance and polymorphism are given in Table 2. Table 2 shows that below 10% of classes of the selected multimedia software are involved in inheritance and polymorphism except VLC where approximately 45% of classes are involved in inheritance and polymorphism. It means a majority of the multimedia software consists of little usage of inheritance and polymorphism. Hence, inheritance and polymorphism play a little role in fault prediction in the case of multimedia software compared to general software. Although complexity, coupling, and cohesion play a significant role in fault prediction of multimedia software. So, more weightage is assigned to TCC, CAC, and LCOCC. Details and usage of all five metrics are shown in Table 4 of the result section.

A comparable process is used to create datasets based on existing code metrics, as the proposed model is evaluated against the existing model. Code metrics used to develop this model are shown in Table 3 of the result section.

3.2 Identification of Threshold Values

This is the second phase of the proposed fault prediction model, which decides the threshold values of our code metrics and connectors used to connect decided threshold values using the Genetic algorithm. Suppose there is a training dataset $X \in \mathbb{R}^{n*d} = \{x_i \in \mathbb{R}^d\}_{i=1}^n$ along with training labels $y = \{0, 1\}^n$, where n represents the number of training samples and d represents the number of dimensions/code metrics. Figure 2 represents the structure of the chromosome while applying the Genetic algorithm to find the optimal threshold values.

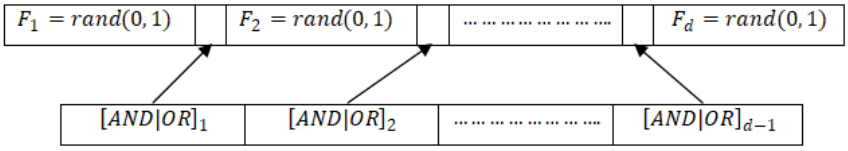

Figure 2: Structure of the Chromosome to apply Genetic Algorithm

In Figure 2, F_i represents the i^{th} element of the chromosome containing any random value between 0 and 1 corresponds to the threshold value of the i^{th} code metric in the training dataset. The size of the chromosome is d, depending upon the total number of

dimensions/code metrics in the dataset. While calculating the fitness (AUC value) of the chromosome by using chromosome elements as thresholds of the code metrics connectors [AND | OR] are inserted randomly in each pair of threshold values. So, total connectors are $\mathrm{d}-1$ in Figure 2 where the i^{th} connector is inserted between i^{th} and $(\mathrm{i}+1)^{\mathrm{th}}$ element of the chromosome while finding the optimal training AUC.

From the six selected multimedia software datasets, one dataset is selected for testing, while the other five are combined to form a training dataset on which Algorithm 1 calculates the metrics threshold values. Notice that the process is repeated five times for each selected dataset, and all the datasets are selected for testing turn by turn. Algorithm 1 presents the overall working of the threshold detection model using a genetic algorithm.

Algorithm 1: Algorithm to find threshold values with connectors using Genetic algorithm

Input:
Training dataset: $X \in \mathbb{R}^{n*d} = \{x_i \in \mathbb{R}^d\}_{i=1}^n$
Training labels:$y = \{0,1\}^n$
Output:
Threshold values: c_{best}
Connectors: o_{best}

Find_fitness_with_operations(X, y, P):

1. $O_{best} = \emptyset, AUC_{best} = \emptyset, O_{size} = 10$
2. *for* C in P:
3. $O^c = generate_random_operations(O_{size}, d-1)$
4. $auc_{best} = 0.5$
5. $O^c_{best} = \emptyset$
6. *for* o^c in O^c:
7. $auc = find_area_under_curve(X, y, C, o^c)$
8. *if*$(auc > auc_{best})$:
9. $auc_{best} = auc$
10. $O^c_{best} = o^c$
11. $O_{best} = append(O^c_{best})$
12. $AUC_{best} = append(auc_best)$
13. $return AUC_{best}, O_{best}$

Find_threshold_values(X, y):

1. $f_{best} = 0.5, c_{best} = \emptyset, o_{best} = \emptyset, C_{rate} = 0.7, M_{rate} = 0.1, epochs = 50, P_{size} = 20$
2. $Population = initialize_random_population(P_{size}, d)$
3. *for* $i = 0$ to $epochs$:
4. $F, O = find_fitness_with_operations(X, y, Population)$
5. $f_{best}, c_{best}, o_{best} = save_best_chromosome(Population, F, O)$
6. $Parent_1, Parent_2 = select_best_chromosomes(Population)$

7. $Offsprings = crossover_population(Population, C_{rate}, Parent_1, Parent_2)$
8. $Offsprings = mutate_population(Offsprings, M_{rate}, Parent_1, Parent_2)$
9. $Population = Offsprings$
10. $return c_{best}, o_{best}$

Algorithm 1 consists of two functions, find_threshold_values() and find_fitness_with_operations(), where function find_threshold_values() runs the genetic algorithm to find the threshold values and function find_fitness_with_operations() finds the fitness value of each chromosome of the population.

Line 1 initializes the parameters of the genetic algorithm, such as population size (P_{size}), maximum iterations ($epochs$), mutation rate (M_{rate}), and crossover rate (C_{rate}), and variables to store best values, such as best fitness (f_{best}), best chromosome (c_{best}), and best-identified connectors (o_{best}) among threshold values. Line 2 generates a random population between 0 and 1, where P_{size} is the population size and d is the dimensions of the generated population. Line 3 is the main loop of the genetic algorithm. Line 4 uses the training dataset to identify the best-suited operation for each chromosome and their best fitness values. Line 5 saves the best-identified fitness value and chromosome along with connectors. Line 6 selects two parents from the population, which are used for crossover and mutation. Line 7 performs crossover on the remaining population based on the selected parents with a 0.7 crossover rate. Line 8 performs mutation on the remaining population with a 0.1 mutation rate. Line 9 assigns the crossover and mutated population to the original population for use in the next iteration of the algorithm while retaining the best chromosome.

Line 4 of the find_threshold_values() function calls find_fitness_with_operations(), which calculates the fitness of each chromosome of the randomly generated population and identifies the connectors to connect the selected threshold values. AUC value is used as the fitness value of the chromosomes, and the object function is the maximization function where AUC must be greater than 0.5. Line 1 creates variables to store the best-identified connectors and the best-achieved AUC values on the training dataset by each chromosome. Line 3 iterates through population P and extracts one chromosome C at a time to find the fitness value. Line 4 generates ten random vectors of connectors for each chromosome C. Lines 5 to 10 test all ten generated vectors one by one on the selected chromosome and identify the best connector vector that achieves the highest AUC value on the training dataset. Each element of the connector vector consists of either AND or OR. If two metrics threshold values are connected with AND, then an instance of the training dataset is considered faulty if the values of both metrics exceed the threshold values. On the other hand, if two metrics threshold values are connected with OR, an instance of the training dataset is considered faulty if any of the two metrics values exceeds the threshold value. After inserting connectors in between, threshold values look like those shown in equation (3).

$$T = TCC_{threshold}[AND|OR]CAC_{threshold}[AND|OR]LCOCC_{threshold}[AND|OR]IMCR_{threshold}[AND|OR]OMCR_{threshold} \quad (3)$$

In equation (3), T is a complete vector of threshold values and connectors inserted between each metric pair. $[AND|OR]$ means either AND or OR connector connects a pair of metrics.

Lines 11 and 12 save the best-identified connector vector along with the achieved AUC value of selected chromosome C. After iterating over the population and storing the best fitness values along with the best-identified connectors of the whole population, Line 13 returns it.

4 Results and Discussions

This section presents the experimentation to decide the threshold value of various code metrics and a way to combine them to better identify faulty classes in multimedia tools. Experimentation is done on a machine consisting of a Corei5 processor and 8GM RAM. Finally, our code metrics-based identification results are compared with those of existing code metrics-based identification and then discussed to identify the better metric suite for fault identification in multimedia tools.

4.1 Selected Software

Table 2 provides details about the multimedia software, such as name, version, components, percentage of classes involved in inheritance, and percentage of classes consisting of polymorphic methods. Threshold values of metrics used in both metrics suites (ours and existing) are decided based on five multimedia software datasets and tested on a sixth dataset turn-by-turn to evaluate the thresholds decided by our proposed technique.

Multimedia software	Version	Components (classes and interface)	Inheritance (%)	Polymorphism (%)
Code tracker master (CTM)	-	147	74.82	9.52
JC Player (JCP)	2.0	19	0.0	5.26
Live video broadcaster (LVB)	-	71	8.45	5.63
MK video player (MKP)	-	110	3.63	0.09
Music player lite (MPL)	-	76	0.0	3.94
VLC media player (VLC)	-	89	46.06	44.94

Table 2: Selected multimedia software

Table 3 presents the details of selected general software datasets to validate the proposed threshold detection technique for general software datasets. These datasets are labelled and openly available in Promise Repository [Karim, 17]. However, to predict the faults in these software datasets our metric suite shown in Table 5 is used

instead of the default code metrics of these datasets because proposed code metrics give better results compared to the default code metrics of these datasets [Singh, 24a] [Singh, 24b].

General software	Version	Components (classes and interface)	Inheritance (%)	Polymorphism (%)
Log4j	1.0	135	21.48	26.66
Log4j	1.2	109	18.34	30.27
Xerces	1.3	453	38.41	16.11
Jedit	4.1	312	45.51	27.88
Camel	1.6	965	46.94	39.79
Ant	1.6	351	54.98	31.33

Table 3: Selected General software

4.2 Code Metrics Details

This section provides an overview of both metrics suites used in this article for fault prediction of multimedia software. Table 3 presents the details, such as metric name, type, and description of the existing metrics. The metric suite shown in Table 3 consists of two coupling metrics, two size metrics, two inheritance metrics, one complexity metric, and five cohesion metrics.

Metric and its type	Name	Description
CBO(Coupling)	Coupling between objects	This metric calculates the total number of incoming and outgoing method calls [Chidamber, 94].
WMC(Complexity)	Weight methods count	This metric calculates the total sum of cyclomatic complexities of the public methods of a class [Chidamber, 94].
DIT(inheritance)	Depth of inheritance	This metric calculates the depth of the class in the inheritance tree [Chidamber, 94].
NOC(Inheritance)	Number of children	This metric calculates the total number of derived classes [Chidamber, 94].
RFC(Coupling)	Response for class	This metric calculates the total method calls in response to the message received by the object of the class [Chidamber, 94].
LCOM(Cohesion)	Lack of cohesion between method pairs	This metric calculates the total number of non-cohesive method pairs in a class [Chidamber, 94].

Coh(Cohesion)	Class cohesion	This metric calculates the accessed field-based cohesion among public methods of the class [Briand, 97].
SCOM(Cohesion)	Structural cohesion metric	This metric calculates the structural cohesion of the class's methods based on the actual parameters of the function call [Ó Cinnéide, 12].
TCC(Cohesion)	Tight class cohesion	This metric is the count of the direct connection in a class [Bieman, 95].
LCC(Cohesion)	Loose class cohesion	This metric is the count of the indirect connections in a class [Bieman, 95].
NOP(Size)	Number of public methods	This metric is the count of the public methods of a class [Bansiya, 02].
LOC(Size)	Line of code	This metric is the count of the total number of lines of code of a class [Tegarden, 95].

Table 4: Existing metrics used for fault identification in multimedia tools

Table 5 presents the details of our metric suite, which consists of one complexity (TCC), one coupling (CAC), one cohesion (LCOCC), one inheritance (IMCR), and one polymorphism (OMCR) metric to cover all object-oriented features because all modern multimedia projects are developed using object-oriented programming languages such as Java. TCC is used over WMC, LOC, and NOP because it utilizes weighted path complexity (WPC) to measure the weights of the methods and also considers data members along with their data types to capture the complexity of the class. CAC is considered over CBO and RFC because it considers function call arguments along with their data types, it also considers return values along with their data types, and this metric utilizes the direction of the function call in the calculation of the coupling, unlike CBO and RFC. LCOCC is chosen over Coh, LCC, SCOM, and LCOM because it considers sequential cohesion through public to private methods call sequence, and private data members along with their data types to capture communicational cohesion. IMCR and OMCR are used to capture special features of object-oriented design instead of DIT and NOC because both of these metrics utilize WPC to quantify the inheritance and polymorphism, unlike DIT and NOC.

Metric and its type	**Name**	**Contribution**	**Description**
TCC(Complexity) [Singh, 24a] [Singh, 24b]	Total class complexity	0.3	This metric calculates the weighted sum of the class members, such as data members, member functions, and inherited methods and variables.

CAC(Coupling) [Singh, 24a] [Singh, 24b]	Coupling among classes	0.3	This metric considers the weights of the formal parameters and returned values captured through function calls in a class.
LCOCC(Cohesion) [Singh, 24a] [Singh, 24b]	Lack of cohesion between connected component pairs	0.3	This metric measures the absence of cohesion between the interconnected components of the class based on the weighted variables, and weights are assigned to the variables based on their data types.
IMCR(Inheritance) [Singh, 24b]	Inherited methods complexity ratio	0.05	The proportion of the inherited methods' complexity to the overall complexity of the class.
OMCR(Polymorphism) [Singh, 24b]	Overloaded methods complexity ratio	0.05	The proportion of the complexity of overloaded methods to the total complexity of the class.

Table 5: Our proposed metrics used for fault identification in multimedia tools

4.3 Results and Discussions

This section provides the results collected based on the experiments done on selected multimedia software. Each experiment is performed five times, and means of AUC and accuracy, along with standard deviation, are collected. Table 6 compares the performance of threshold-based fault prediction using both metrics suites based on accuracy, AUC, and harmonic mean of these two, where the best results are highlighted in bold. When comparing fault prediction performance based on accuracy, in the case of four multimedia software, our metrics suite-based fault prediction using decided threshold values gives better results, and for the remaining two datasets, the existing metrics suite-based model gives better results. However, most of the fault prediction datasets are imbalanced because the developed software consists of very few faulty classes, and accuracy is not a good choice for testing the performance of the developed model for imbalanced datasets. So, the AUC performance metric is used to evaluate both metrics suite-based fault prediction models and our metrics suite-based model performance better for five multimedia software datasets out of six selected multimedia software. The last comparison is based on the harmonic mean (HM) of these two

performance metrics to show the balance. Our metrics suite-based developed model gives better results for five out of six selected multimedia software.

Datasets	ROC-AUC		Accuracy		HM	
	Suggested	Existing	Suggested	Existing	Suggested	Existing
CTM	**0.83430** ±0.0994	0.80588 ±0.0527	**0.81115** ±0.0546	0.80952 ±0.0456	**0.82256**	0.8077
JCP	**0.96875** ±0.0	0.87833 ±0.0817	**0.94737** ±0.0	0.89474 ±0.0526	**0.95794**	0.88646
LVB	**0.89867** ±0.0523	0.84656 ±0.0199	**0.89295** ±0.0077	0.83662 ±0.0380	**0.8958**	0.84156
MKP	**0.91341** ±0.0414	0.875 ±0.0204	0.87818 ±0.0081	**0.90909** ±0.0143	**0.89545**	0.89172
MPL	**0.86979** ±0.0214	0.81958 ±0.0141	**0.82895** ±0.0093	0.79473 ±0.0432	**0.84888**	0.80696
VLC	0.80719 ±0.0609	**0.85761** ±0.0189	0.77977 ±0.0453	**0.80674** ±0.0184	0.79324	**0.8314**

Table 6: Comparison of both metrics suites based on achieved AUC and accuracy on multimedia software

Figure 3 compares the fault prediction performance of both metrics suite-based models using box plots where red box plots show the accuracy and AUC achieved by the existing metrics suite-based model, and blue box plots present the results of our metrics suite-based fault prediction model. Figure 3(a) compares both metrics suites based on accuracy, where our metrics' thresholds-based model performs better for four out of six selected multimedia software, and the accuracy difference is large. Figure 3(b) compares based on the achieved AUC value on selected multimedia software, and the proposed model performs better for five out of six datasets. Circles on both sides of the box plots show the outliers, and circles on the negative side of the box plots (negative side) are undesirable.

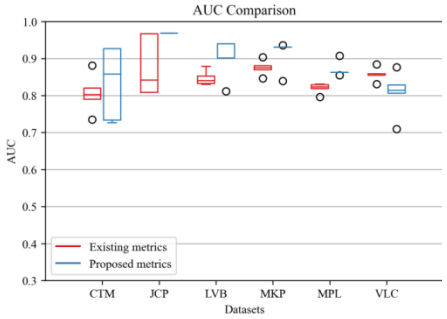

Figure 3(a): AUC comparison

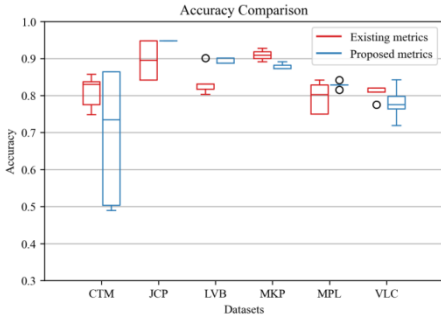

Figure 3(b): Accuracy comparison

Figure 3: Comparison of both metrics suites based on AUC and accuracy using multimedia software

Figure 4 compares both techniques based on the harmonic mean of both performance metrics using a bar graph, which shows the balance between these two performance metrics. In Figure 4, blue bars show the performance achieved by the threshold-based model of existing metrics, and red bars show the performance of our model. In the case of five multimedia tools' datasets, the proposed technique performs better than the existing technique, and the performance difference is significant. For VLC software, the existing metrics' threshold-based model gives better performance.

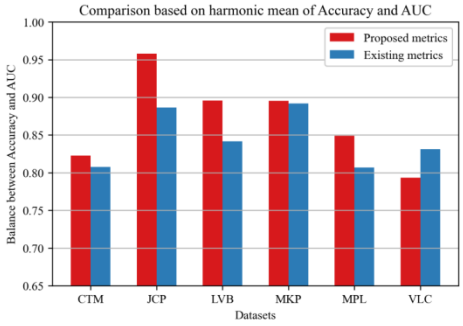

Figure 4: Comparison based on harmonic mean (HM)

Based on Table 6 and Figures 3 and 4, it can beinferred that the proposed model is more suitable than existing models for fault prediction on multimedia software because it performs better with fewer metrics.

Table 7 presents the threshold values of our code metrics, shown in Table 5, along with connectors. The experiment is repeated five times on all selected multimedia software datasets and final threshold values are decided by taking the mean and median of each run on all selected software. Both mean and median threshold values of all runs are reported in Table 7 for each code metric shown in Table 5. Connectors among these code metrics are decided based on the mode of all five runs. The last column of Table 7 shows the difference between the median and mean threshold values of all runs. It can be inferred from Table 7 that the decided threshold values by the Genetic algorithm for each code metric are almost equal for all runs on all selected software datasets because difference between median and mean is very little and has no outlier. The symbol '#' shows the connector between two code metrics' threshold values in Table 7.

Metric	**Thresholds**			**Concatenation** i^{th} # $(i+1)^{th}$
	Mean	**Median**	**Diff.**	
TCC	0.664	0.648	0.016	TCC *OR* CAC
CAC	0.646	0.678	0.031	CAC *AND* LCOCC
LCOCC	0.326	0.310	0.015	LCOCC *OR* IMCR
IMCR	0.490	0.436	0.053	IMCR *OR* OMCR
OMCR	0.253	0.234	0.018	-

Table 7: Decided threshold values for new metrics suite and their concatenation for selected multimedia software

Table 8 presents the mean and median of the threshold values decided for existing code metrics, along with connectors among these metrics based on the experiments performed on selected multimedia tools' datasets. The last column of Table 8 presents the difference between the mean and median of the identified threshold values, and the maximum difference is 5% between the mean and median for these metrics except DIT and COH, where the difference is 8%. Overall, it can be concluded that the threshold values of all metrics are almost equivalent for all selected software datasets. The connectors among these code metrics are decided by taking the mode of all runs on selected software datasets.The operator '#' in Table 8 shows the connector between two code metrics threshold values.

Metric	**Thresholds**			**Concatenation**
	Mean	**Median**	**Diff.**	$\mathbf{i^{th}}$ **#** $\mathbf{(i+1)^{th}}$ **metric**
CBO	0.54338	0.5101	0.03328	CBO OR WMC
WMC	0.44032	0.43218	0.00814	WMC OR DIT
DIT	0.59979	0.67981	0.08002	DIT AND NOC
NOC	0.48939	0.44256	0.04683	NOC AND RFC
RFC	0.63378	0.6578	0.02402	RFC OR LCOM
LCOM	0.32687	0.33008	0.00321	LCOM OR COH
COH	0.38081	0.29825	0.08256	COH AND SCOM
SCOM	0.46806	0.46587	0.00219	SCOM AND TCC
TCC	0.51586	0.50977	0.00609	TCC AND LCC
LCC	0.40948	0.38946	0.02002	LCC AND NOP
NOP	0.55923	0.61008	0.05085	NOP OR LCOM
LOC	0.58938	0.61536	0.02598	-

Table 8: Decided threshold values for existing metrics suite and their concatenation for selected multimedia software

The prediction performance on six general software datasets is also shown to validate the effectiveness of the decided threshold values of both metrics suites based on genetic algorithm. Table 9 compares both models based on the collected AUC values on selected six general software datasets based on the mean and median of the threshold values. Notice that the collected AUC based on mean and median is the same, which means the decided threshold values of the metrics are almost equal for all selected software datasets for both models. The proposed model either gives a better AUC value or a comparable value to the existing model on selected general software datasets, whereas the existing model completely fails on Log4j-1.2.

Datasets	**AUC Proposed**		**AUC Existing**	
	Mean	**Median**	**Mean**	**Median**
Log4j-1.0	**0.75556**	**0.75556**	0.74815	0.74815
Log4j-1.2	**0.66972**	**0.66972**	0.10084	0.10084
Xerces-1.3	0.84768	0.84768	**0.84951**	**0.84466**
Jedit-4.1	0.74679	0.74679	**0.7637**	**0.7637**
Camel-1.6	**0.80207**	**0.80207**	0.79036	0.79036
Ant-1.6	**0.73789**	**0.73789**	0.73547	0.73547

Table 9: comparison of decided threshold values on general software

Figure 5 compares both models using a bar graph where red bars show the performance achieved by the proposed technique and blue bars achieved by the existing technique. The proposed technique perfectly works on all six software datasets; conversely, the existing technique fails on Log4j-1.2. Figure 5(a) shows the results collected based on the mean of the threshold values, and Figure 5(b) shows the results collected based on the median of the thresholds.

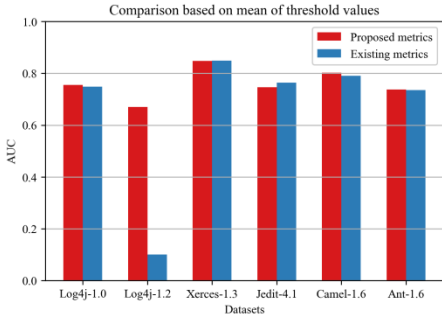

Figure 5(a): Mean of threshold values

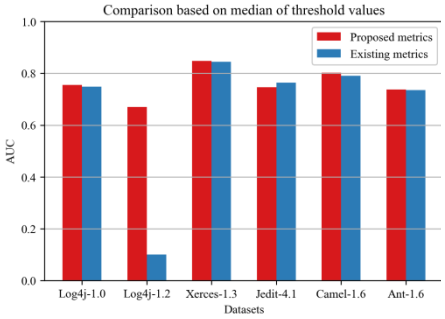

Figure 5(b): Median of threshold values

Figure 5: AUC comparison of metrics suits based on mean and median of thresholds

Based on Table 9 and Figure 5, it can be concluded that the threshold values calculated based on our proposed technique are universal, and the proposed model works excellent for multimedia software and gives satisfactory results for general software, whereas existing models fail in some situations.

4.4 Comparison with Existing Studies

This section compares the proposed technique with ROC-based threshold detection and VARL threshold detection techniques Bender et al. [Bender, 99]Shatnawi et al. [Shatnawi, 10a] based on AUC using both multimedia and general datasets.

Table 10 compares the proposed technique with ROC and VARL based on AUC using multimedia software datasets shown in Table 2. The best AUC values in Table 10 are shown in bold letters. The proposed technique-based threshold values give significantly better results than ROC and VARL in the case of all six multimedia software datasets. In the case of ROC and VARL, our code metrics have been used and threshold values are combined with the AND operator. It means if any of the selected code metrics' threshold value is violated then the class is considered faulty. On the other hand, the proposed technique finds collective threshold values based on the Genetic algorithm and simultaneously finds the operators to combine decided threshold values for fault prediction. This section utilizes our proposed code metrics to compare the proposed technique with ROC and VARL because our code metrics give better performance than existing code metrics as proved based on Table 6 and Table 9.

Datasets	ROC	VARL	Proposed
CTM	0.522	0.501	**0.83430**
JCP	0.515	0.5	**0.96875**

LVB	0.5	0.5	**0.89867**
MKP	0.612	0.592	**0.91341**
MPL	0.531	0.513	**0.86979**
VLC	0.532	0.511	**0.80719**

Table 10: Comparison with previous research based on multimedia software

Table 11 compares the proposed technique with ROC and VARL based on general software datasets openly available in the Promise repository [Karim, 17] using the AUC performance metric. Results shown in Table 11 are collected using our code metrics, and the best values are highlighted in bold. Clearly, in the case of all six general software datasets, the proposed technique gives significantly better results than both ROC and VARL methods.

Datasets	**ROC**	**VARL**	**Proposed**
Log4j-1.0	0.61	0.59	**0.75556**
Log4j-1.2	0.5	0.5	**0.66972**
Xerces-1.3	0.59	0.573	**0.84768**
Jedit-4.1	0.553	0.511	**0.74679**
Camel-1.6	0.514	0.5	**0.80207**
Ant-1.6	0.573	0.53	**0.73789**

Table 11: Comparison with previous research based on General software datasets

Figure 6 compares the proposed technique with ROC and VARL where green box plots show the AUC achieved by the proposed technique, red box plots show the AUC of ROC, and blue box plots show the AUC of VARL-based threshold values. Further, the first group shows the AUC comparison on multimedia software datasets and the second group shows the AUC comparison on general software datasets. Clearly, the median lines of green box plots are higher than blue and red box plots in their respective groups. So, it can be inferred that the proposed technique-based threshold values give better SFP performance than both ROC and VARL for both multimedia and general software datasets. Further, if green box plots of multimedia and general software datasets are compared then it can be inferred that the performance slightly deteriorates for general software datasets compared to multimedia software datasets. However, the proposed technique is significantly better than the compared techniques for both multimedia and general software datasets.

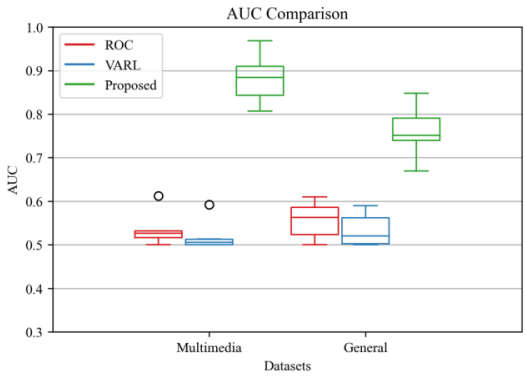

Figure 6: Comparison of proposed technique with VARL and ROC

5 Conclusions

This research article proposes a new fault prediction model based on metrics thresholds for multimedia software to reduce testing time and effort. The proposed model works in two phases. The first phase of the algorithm generates the unlabelled dataset by extracting structural code metrics of existing multimedia software. Then, it scales the metrics by giving more weightage to complexity, coupling, and cohesion metrics and less weight to inheritance and polymorphism metrics to change the contribution of structural aspects for fault prediction in multimedia software, and then the scaled dataset is labeled by finding similar classes from the other software datasets. The second phase decides the thresholds of the code metrics and identifies the connectors among thresholds based on genetic algorithm using AUC as fitness function. The proposed SFP technique is compared with the existing techniques using ROC-AUC and accuracy performance metrics, and results prove that our code metrics' threshold values work far better than existing metrics' threshold values for multimedia software and give satisfactory results for general software with fewer code metrics than the existing metric suite. However, the proposed work has some limitations such as the proposed technique is tested on small to medium-scale multimedia software datasets and can't be generalized for all software datasets. Similarly, machine learning-based prediction may be more accurate, if a suitable dataset of good size is available. In general, the proposed threshold detection technique is quite useful for any kind of software to find the project-specific objective-based threshold values and it finds the collective thresholds that are better than existing techniques-based threshold values.

6 Future Work

In the future, we will further increase the performance of the metrics' threshold-based model by exploring it better using advanced optimizers. As the threshold values are project-dependent and selection of projects is highly important to decide the effective threshold values. So, a project selection procedure can be developed to decide the

effective threshold values. Further, work can be extended to find multiple threshold values, through which the severity of the fault can be predicted.

References

[Alves, 10] Alves, Tiago L., Christiaan Ypma, and Joost Visser.: "Deriving metric thresholds from benchmark data." In 2010 IEEE international conference on software maintenance, (2010), 1-10.

[Arar, 16] Arar, Ömer Faruk, and Kürşat Ayan.: "Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies." Expert Systems with Applications, 61 (2016), 106-121.

[Bansiya, 02] Bansiya, Jagdish, and Carl G. Davis.: "A hierarchical model for object-oriented design quality assessment." IEEE Transactions on software engineering, 28, 1 (2002), 4-17.

[Bender, 99] Bender, Ralf.: "Quantitative risk assessment in epidemiological studies investigating threshold effects." Biometrical Journal: Journal of Mathematical Methods in Biosciences, 41, 3 (1999), 305-319.

[Bieman, 95] Bieman, James M., and Byung-Kyoo Kang.: "Cohesion and reuse in an object-oriented system." ACM SIGSOFT Software Engineering Notes ,20, SI (1995), 259-262.

[Bishnu, 11] Bishnu, Partha S., and Vandana Bhattacherjee.: "Software fault prediction using quad tree-based k-means clustering algorithm." IEEE Transactions on knowledge and data engineering, 24, 6 (2011), 1146-1150.

[Boucher, 18] Boucher, Alexandre, and Mourad Badri.: "Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison." Information and Software Technology, 96 (2018), 38-67.

[Briand, 97] Briand, Lionel, Prem Devanbu, and Walcelio Melo.: "An investigation into coupling measures for C++." In Proceedings of the 19th international conference on Software engineering, (1997), 412-421.

[Catal, 09] Catal, C., Sevim, U., &Diri, B. :"Clustering and metrics thresholds based software fault prediction of unlabeled program modules." In 2009 Sixth international conference on information technology: new generations, (2009, April), 199-204).

[Catal, 09] Catal, Cagatay, Ugur Sevim, and Banu Diri.: "Clustering and metrics thresholds based software fault prediction of unlabeled program modules." In 2009 Sixth international conference on information technology: new generations IEEE, (2009),199-204.

[Catal, 11] Catal, Cagatay, Oral Alan, and Kerime Balkan.: "Class noise detection based on software metrics and ROC curves." Information Sciences, 181, 21 (2011), 4867-4877.

[Chidamber, 94] Chidamber, Shyam R., and Chris F. Kemerer.: "A metrics suite for object oriented design." IEEE Transactions on software engineering, 20, 6 (1994), 476-493.

[Colakoglu, 21] Colakoglu, F. N., Yazici, A., & Mishra, A.: "Software product quality metrics: A systematic mapping study." IEEE Access, 9 (2021), 44647-44670.

[Dejaeger, 12] Dejaeger, Karel, Thomas Verbraken, and Bart Baesens.: "Toward comprehensible software fault prediction models using bayesian network classifiers." IEEE Transactions on Software Engineering, 39, 2 (2012), 237-257.

[Gyimóthy, 05] Gyimóthy, Tibor, Rudolf Ferenc, and Istvan Siket.: "Empirical validation of object-oriented metrics on open source software for fault prediction." IEEE Transactions on Software engineering, 31, 10 (2005), 897-910.

[Kanwar, 23] Kanwar, Shailza, Lalit Kumar Awasthi, and Vivek Shrivastava.: "Candidate project selection in cross project defect prediction using hybrid method." Expert Systems with Applications, 218 (2023), 119625.

[Karim, 17] Karim, S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B.: "Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset." In 2017 IEEE international conference on cybernetics and computational intelligence, (2017, November), 19-23.

[Liu, 23] Liu, Jingyu, Jun Ai, Minyan Lu, Jie Wang, and Haoxiang Shi.: "Semantic feature learning for software defect prediction from source code and external knowledge." Journal of Systems and Software, (2023), 111753.

[Malhotra, 15] Malhotra, Ruchika, and Ankita Jain Bansal.: "Fault prediction considering threshold effects of object-oriented metrics." Expert Systems, 32, 2 (2015), 203-219.

[McCabe, 76] McCabe, Thomas J.: "A complexity measure." IEEE Transactions on software Engineering, 4 (1976), 308-320.

[Mishra, 21] Mishra, A., Shatnawi, R., Catal, C., & Akbulut, A.: "Techniques for calculating software product metrics threshold values: a systematic mapping study." Applied Sciences, 11, 23 (2021), 11377.

[Ó Cinnéide, 12] Ó Cinnéide, Mel, Laurence Tratt, Mark Harman, Steve Counsell, and Iman Hemati Moghadam.: "Experimental assessment of software metrics using automated refactoring." In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, (2012), 49-58.

[Pachouly, 22] Pachouly, Jalaj, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, and Ajith Abraham.: "A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools." Engineering Applications of Artificial Intelligence, 111 (2022), 104773.

[Prajapati, 22] Prajapati, Amarjeet, Anshu Parashar, Sunita, Jitender Kumar Chhabra, and Chakresh Kumar Jain.: "Multimedia in search-based software engineering: challenges and opportunities within a new research domain." Multimedia Tools and Applications 81, 25 (2022), 35671-35691.

[Rathee, 19] Rathee, Amit, and Jitender Kumar Chhabra.: "Reusability in multimedia softwares using structural and lexical dependencies." Multimedia Tools and Applications, 78, 14 (2019), 20065-20086.

[Rathee, 22] Rathee, Amit, and Jitender Kumar Chhabra.: "Feature-based critical components identification in multimedia software." Multimedia Tools and Applications, 81, 25 (2022), 35595-35618.

[Rosenberg, 98] Rosenberg, Linda.: "Applying and interpreting object oriented metrics." In Software technology conference, Utah, (1998).

[Sharma, 23] Sharma, Umamaheswara, and Ravichandra Sadam.: "How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model." Journal of Systems and Software, 195 (2023), 111522.

[Sharma, 24] Sharma, D., & Chandra, P.: "An empirical analysis of software fault proneness using factor analysis with regression." Multimedia Tools and Applications, 83, 17 (2024), 52535-52591.

[Shatnawi, 10a] Shatnawi, Raed, Wei Li, James Swain, and Tim Newman.: "Finding software metrics threshold values using ROC curves." Journal of software maintenance and evolution: Research and practice, 22, 1 (2010), 1-16.

[Shatnawi, 10b] Shatnawi, Raed.: "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems." IEEE Transactions on software engineering, 36, 2 (2010), 216-225.

[Singh, 14] Singh, Satwinder, and K. S. Kahlon.: "Object oriented software metrics threshold values at quantitative acceptable risk level." CSI transactions on ICT, 2 (2014), 191-205.

[Singh, 23] Singh, Manpreet, and Jitender Kumar Chhabra.: "Improved Software Fault Prediction Model Based on Optimal Features Set and Threshold Values Using Metaheuristic Approach." SN Computer Science, 4, 6 (2023), 770.

[Singh, 24] Singh, Manpreet, and Jitender Kumar Chhabra.: "Improved software fault prediction using new code metrics and machine learning algorithms." Journal of Computer Languages, 78 (2024), 101253.

[Singh, 24a] Singh, M., & Chhabra, J. K.: "Improved software fault prediction using new code metrics and machine learning algorithms." Journal of Computer Languages, 78 (2024), 101253.

[Singh, 24b] Singh, M., & Chhabra, J. K.: "Machine learning based improved cross-project software defect prediction using new structural features in object oriented software." Applied Soft Computing, (2024)112082.

[Tang, 24] Tang, Yu, Qi Dai, Ye Du, Lifang Chen, and Xuanwen Niu.: "A software defect prediction method based on learnable three-line hybrid feature fusion." Expert Systems with Applications, 239 (2024), 122409.

[Tegarden, 95] Tegarden, David P., Steven D. Sheetz, and David E. Monarchi.: "A software complexity model of object-oriented systems." Decision Support Systems, 13, 3-4 (1995), 241-262.

[Yu, 12] Yu, L., & Mishra, A.: "Experience in predicting fault-prone software modules using complexity metrics." Quality Technology & Quantitative Management, 9, 4 (2012), 421-434.

[Zhou, 06] Zhou, Yuming, and Hareton Leung.: "Empirical analysis of object-oriented design metrics for predicting high and low severity faults." IEEE Transactions on software engineering, 32, 10 (2006), 771-789.