



Predictive software maintenance utilizing cross-project data

Yogita Khatri¹ · Sandeep Kumar Singh¹

Received: 22 September 2022 / Revised: 16 May 2023 / Accepted: 24 May 2023 / Published online: 23 June 2023

© The Author(s) under exclusive licence to The Society for Reliability Engineering, Quality and Operations Management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden 2023

Abstract To improve the software quality and reduce the maintenance cost, cross-project fault prediction (CPFP) identifies faulty software components in a particular project (aka target project) using the historical fault data of other projects (aka source/reference projects). Although several diverse approaches/models have been proposed in the past, there exists room for improvement in the prediction performance. Further, they did not consider effort-based evaluation metrics (EBEMs), which are important to ensure the model's application in the industry, undertaking a realistic constraint of having a limited inspection effort. Besides, they validated their respective approaches using a limited number of datasets. Addressing these issues, we propose an improved CPFP model with its validation on a large corpus of data containing 62 datasets in terms of EBEMs (PIM@20%, Cost-effectiveness@20%, and IFA) and other machine learning-based evaluation metrics (MLBEMs) like PF, G-measure, and MCC. The reference data and the target data are first normalized to reduce the distribution divergence between them and then the relevant training data is selected from the reference data using the KNN algorithm. Seeing the experimental and statistical test results, we claim the efficacy of our proposed model over state-of-the-art CPFP models namely the Turhan-Filter and Cruz model comprehensively. Thus, the proposed CPFP model provides an effective solution for

predicting faulty software components, enabling practitioners in developing quality software with lesser maintenance cost.

Keywords Cross-project fault prediction · Machine learning-based evaluation metrics · Effort-based evaluation metrics · Software quality · Software maintenance

1 Introduction

The size and the complexity of the software are growing exponentially with growing needs and technical challenges, which can lead to faults, causing a substantial loss in terms of money and time. Thus, developing quality software demands a huge investment in terms of time and money (Arar and Ayan 2015). Further, the power of traditional testing approaches can never always be guaranteed (Kassab et al. 2017). Besides, the delayed identification of faults can increase their fixing cost by 100%, if detected after the development phase (Pelayo and Dick 2007). Therefore, it becomes extremely important to identify the faulty software components before the software release, not only to improve its quality but also to reduce its maintenance cost.

Machine learning techniques are becoming extremely popular in predicting faulty software components so that corrective measures should be taken on time to prevent their failures and reduce their maintenance cost (Bowes et al. 2018; D'Ambros et al. 2012; Lessmann et al. 2008; Menzies et al. 2007). However, the success of a machine learning-based technique largely depends on the amount of potential data. Thus, when sufficient within-project historical fault/defect data is available for training and validating the model, we term it within-project fault prediction (WPPF). A good amount of work (Bisi and Goyal 2016; Canfora et al. 2015;

✉ Yogita Khatri
19403019@mail.jiit.ac.in

Sandeep Kumar Singh
sandeepk.singh@jiit.ac.in

¹ Department of Computer Science Engineering and Information Technology, Jaypee Institute of Information Technology, Noida 201309, India

Liu et al. 2010; Lu et al. 2012; Wang et al. 2015) has been reported in the existing literature on WFPF, proposing various effective approaches. However, in reality, collecting this fault data is an extremely time-consuming and costly task. Thus, under the situation of building a new software project, for which either no historical fault data is available or a very less amount of data is available, WFPF may not provide an efficient solution. However, several software repositories like PROMISE and NASA contain the historical fault data of various open-source projects, which can be leveraged to construct a fault prediction model for a specific software project (aka target project) facing the scarcity of the within-project training data. Thus, when constructing a fault prediction model for a software project (having either limited or no within-project training data) leveraging the fault data of some other project(s) (aka reference/source project (s)), we term it cross-project fault prediction (CPFP). The reference and the target data may or may not have the same feature space. However, we particularly have focussed on building a CPFP model where both target and the reference data share the same feature space. Even if the target and the reference projects have the same set of features, they may have differences in the distribution of their features.

Due to these differences in their distributions, the machine learning model's prediction performance deteriorates. This happens because generally machine learning algorithms are based on the rationale of utilizing similar distribution data in their construction and validation phases. Several researchers have proposed different alternative solutions (Chen et al. 2015; Herbold 2013; Kawata et al. 2015; Ma et al. 2012; Nam et al. 2013; Ryu et al. 2017; Turhan et al. 2009) to lessen the distribution difference between the reference project and the target project data, but there still exists room for the improvement in the prediction performance. Further, majority of them considered only standard machine learning-based evaluation metrics (MLBEMs) like recall, precision, G-mean, F-measure, G-measure, balance, AUC (area under the curve), MCC (Mathew's correlation coefficient), PF (probability of false alarm), etc. to assess their model's potential. They did not consider the effort-based evaluation metrics (EBEMs) under the practical constraint of limited inspection resources, which accounts for the model's applicability in a real scenario. Besides, they validated their approaches using a limited number of selected datasets.

Targeting these research gaps, we aim to propose a machine learning-based improved CPFP model, considering its holistic evaluation using EBEMs along with MLBEMs on a large corpus of 62 datasets. Our proposed approach consists of three phases namely the normalization phase, training data selection phase, and model construction phase. In the normalization phase, the reference data and the target data are normalized to lessen the distribution dissimilarity between them. In the training data selection phase, the

relevant data is selected from the available reference data leveraging the KNN algorithm. The third phase constitutes building the CPFP model using the selected reference data retrieved from the previous phase, with its validation using the normalized target data retrieved from the first phase. To measure the true performance of the proposed CPFP model, we compare it with state-of-the-art approaches namely, Turhan-Filter (2009) and Cruz model (2009) using 62 datasets from the PROMISE repository.

On the basis of the observed results, the study makes the following contributions to the CPFP literature:

1. We propose an improved machine learning-based CPFP model and validate its performance in terms of EBEMs and MLBEMs under the realistic scenario of a limited inspection effort, using 62 datasets from the PROMISE repository.
2. The proposed model showcases its effectiveness in terms of MLBEMs by satisfying the practitioner's expectation to balance between the accuracy of both faulty and non-faulty classes over state-of-the-art approaches Turhan-Filter and Cruz model by effectively reducing the distribution gap between the target the reference data.
3. Further, considering the realistic constraint of limited inspection effort, the proposed model justifies its application into practice by better balancing between its cost-effectiveness and the amount of developer's effort required to investigate the faulty modules due to the effective selection of the training data.

The rest of the paper is arranged as follows: Sect. 2 puts light on the existing literature, and Sect. 3 explains the proposed approach in detail. Further ahead, Sect. 4 describes the datasets, evaluation metrics, research questions, and the experimental design of the proposed approach, whereas Sect. 5 describes the experimental results, its analysis, and the ablation study to investigate the contribution of the normalization phase and the training data selection phase in the final performance of the proposed approach. Next, Sect. 6 presents the validity threats with Sect. 7 summarizing the research work along with the future directions.

2 Related work

To the best of our knowledge, Briand et al. (2002) made the earliest contribution to exploring the feasibility of CPFP. Zimmermann et al. (2009) also attempted to explore the feasibility of CPFP by performing a big-scale experimental study. But the experimental results were disappointing. The prime reason for the unsatisfactory performance of CPFP was the distribution gap in the training and testing data.

Addressing this concern, Watanabe et al. (2008) explored the application of the data standardization technique to lessen the divergence in the reference and the target data distributions. Similarly, Cruz and Ochimizu (2009) attempted to normalize the reference and the target data to minimize the difference in their distributions. Both obtained better results, but it was still not satisfactory. Moreover, they evaluated their respective approaches on limited datasets using MLBEMs only and did not pay any attention to EBEMs.

Further, Turhan et al. (2009) explored a training data selection method, where they selected 10 matching software modules from the reference data for each target data module by applying the KNN algorithm. They achieved improvement in the prediction performance, particularly with respect to recall and PF. Unfortunately, PF was still very high. In addition, the evaluation in terms of EBEMs was not performed.

Working on similar lines, transfer learning was explored by Ma et al. (2012) to bridge the gap across the reference and the target data distributions. The training data were assigned weights using the Data Gravitation method (Peng et al. 2009) and Bayes logic was incorporated to develop the CFPF model. They recorded the improvement with respect to F-measure, PF, and AUC in comparison to conventional CFPF (a CFPF model built directly using the reference data without any processing) and Turhan-Filter (2009). However, the evaluation in terms of EBEMs was not performed.

To further ameliorate the prediction performance, another transfer learning-based technique called TCA+ (transfer component analysis (Pan et al. 2011)) was explored by Nam et al. (2013). They evaluated their model on 8 datasets and witnessed the improvement with respect to the mean F-measure score over conventional CFPF and WFPF.

The above approaches primarily worked towards improving recall. As a result, they observed high PF. A higher false positive rate can ruin the developer's effort and time. Addressing this issue, Chen et al. (2015) proposed a DTB approach, wherein they first assigned weights to the training data and then constructed an ensemble model by applying the Transfer AdaBoost algorithm, leveraging 10% of the within-project labeled data. They tested their approach on 15 datasets and witnessed its supremacy over compared approaches, particularly with respect to PF, G-measure, and MCC. However, the evaluation in terms of EBEMs was completely ignored.

Similar to the DTB approach, Ryu et al. (2017) came up with another transfer boosting-based approach called TCSBoost, which considered class imbalance and feature distribution to assign weights to the training modules and witnessed improvement in the performance with respect to G-mean and balance. Like previous approaches, they also missed considering the evaluation in terms of EBEMs.

All the above CFPF approaches evaluated their respective models in terms of MLBEMs only and considered only some selected datasets from the openly available pool of datasets.

Further, different approaches used different datasets and diverse evaluation metrics, one cannot specify which technique performed the best comprehensively. Therefore, to find out the best CFPF approach, Herbold et al. (2018) carried out a large empirical study and compared 24 existing CFPF approaches and concluded the Cruz model (2009) and the Turhan Filter (2009) as the top two CFPF approaches.

In an attempt to further improve the prediction performance comprehensively, the study proposes an improved CFPF model inspired by Cruz and Ochimizu (2009), Turhan et al. (2009) with its evaluation on 62 datasets in terms of EBEMs along with MLBEMs and its comparison with the two best state-of-the-art CFPF approaches Cruz model (2009) and the Turhan-Filter (2009).

3 Proposed approach

Figure 1 depicts the framework of the proposed model. It contains three main phases. The very first phase is the normalization phase. Following the Cruz approach (2009), we first normalize the reference data and the target data to narrow the distribution gap between them. But different from the Cruz approach which used only a single project data for training, we use multiple project data in the training phase and consider the median of the target data as the reference point to normalize the reference and the target data. Kindly note that the reference data contains the data of all available projects except the target data and its related version data. After normalization, our second phase starts, which basically works on filtering the relevant data from the available reference data. In particular, we eliminate the irrelevant software modules from the reference data by selecting the 10 nearest software modules from the reference data, for each software module in the target data. This phase helps us in selecting the reference data which is similar to the target data. Further ahead, the third phase constitutes the construction of a CFPF model using the reference data retrieved from the second phase with its validation on the normalized target data.

To have a better understanding of the model proposed, we also have presented the flow of model building through a flowchart as shown in Fig. 2.

3.1 Normalization phase

As shown in Fig. 2, this phase mainly works towards decreasing the gap between the reference and the target data distributions. Following the Cruz approach (2009), we normalized both the reference as well as the target data.

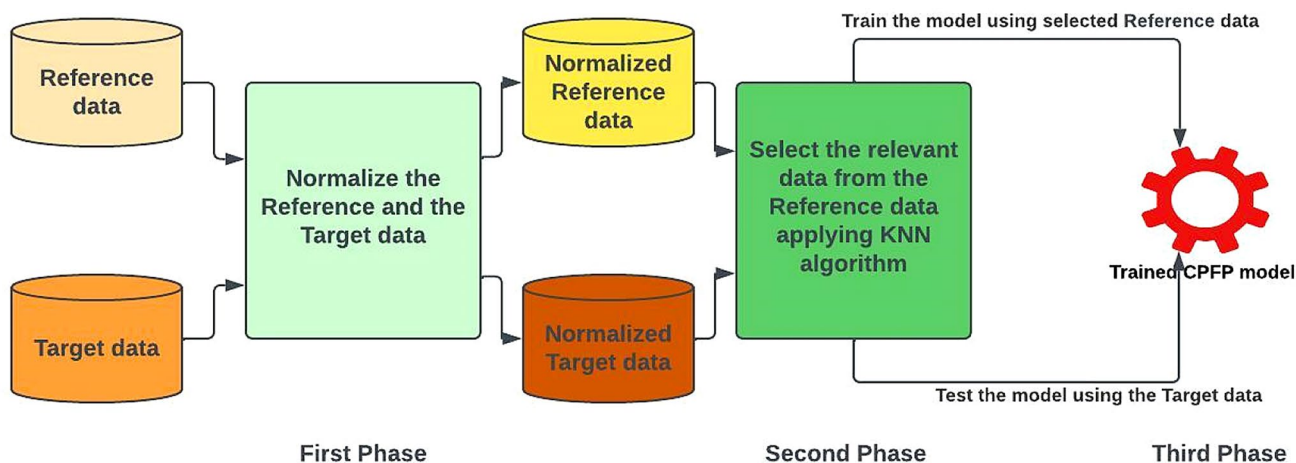


Fig. 1 Proposed CFPF model’s framework

But they used the data of only one project for training and considered the median of that data as the reference point. Different from them, our reference data contains multiple

project data, therefore as a reference point, we considered the median of the target data. In other words, we used log transformation and the median of the target data as the reference point to normalize the reference and the target data using the following equation.

$$M'_i = \log(1 + M_i) + (\text{median}(\log(1 + M_i)) - \text{median}(\log(1 + M_i^T)))$$

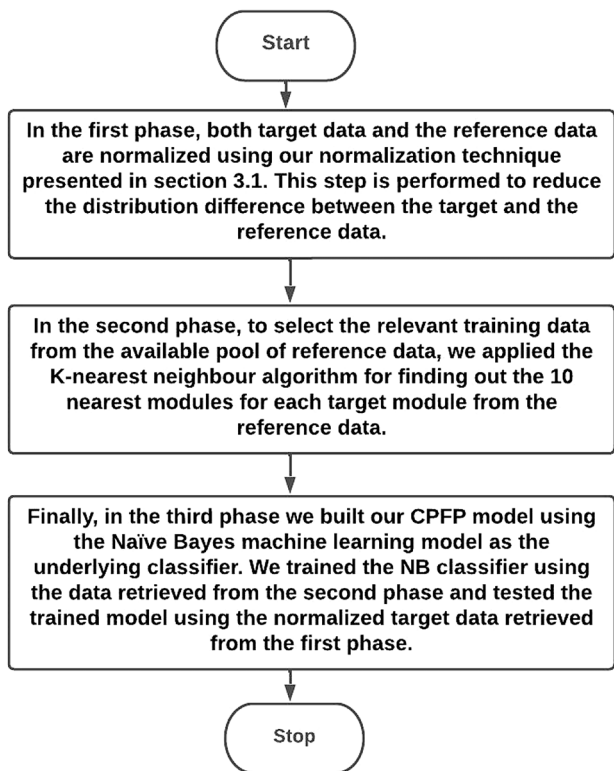


Fig. 2 Proposed model’s building steps

where M'_i shows the normalized value of the i^{th} metric M_i and M_i^T represents the same i^{th} metric from the target data.

3.2 Training data selection phase

This phase worked on selecting the reference data matching with the target data. For each target data module, 10 nearest modules (considering the Euclidean distance) were selected from the reference data by applying the KNN algorithm (Cover and Hart 1967). The selected data was then searched for any duplicate modules. If found, they were then deleted from the selected reference data.

3.3 Model construction phase

The third phase worked on building the CFPF model using the reference data retrieved from the previous phase. We used Naïve Bayes (NB) as the underlying machine learning classifier for our proposed CFPF model. This is because NB has proved its potential in many previous studies in comparison to other machine learning models (Chen et al. 2015; He et al. 2015; Lessmann et al. 2008; Menzies et al. 2007; Ryu et al. 2017) and is used the most across existing CFPF studies (Hosseini et al. 2019; Khatri

and Singh 2021). Next, we validated its potential using the normalized target data and calculated the MLBEMs and EBEMs. Kindly note that the detailed explanation of these evaluation metrics and the process of their calculation is explained in Sect. 4.2.

4 Datasets, performance measures and experimental design

This section describes the datasets, evaluation metrics, research question, baseline selection and experimental design of the proposed approach.

4.1 Datasets

For evaluating the potential of our proposed approach, 62 fault datasets from the PROMISE repository are utilized. These datasets consist of 15 closed-source project datasets and 47 versions of 17 open-source project datasets. These datasets were collected by Jureczko and Madeyski (2010). Each dataset consists of 20 static code metrics and the fault count. These software metrics represent the features of the dataset and the fault count represents the fault-proneness of the software module. A fault count with '0' value depicts a non-faulty module, whereas a fault count greater than or equal to 1 shows a faulty module. Table 1 shows the detailed characteristics of these datasets. The first, second, and the third columns represent the short notation, name, and description of each dataset respectively. The fourth and the fifth columns depict the module count and the percentage of faulty modules in each dataset respectively.

We particularly selected a large pool of these datasets including both open-source and closed-source projects to counter the limitation of the many existing studies (Chen et al. 2015; Cruz and Ochimizu 2009; Ma et al. 2012; Nam et al. 2013; Ryu et al. 2017; Watanabe et al. 2008) as they used very limited datasets, which mainly consisted of open-source project datasets. Further, these datasets cover both small to large software projects containing modules ranging from 10 to 965 with fault rates varying from 2.24 to 92.19. A detailed description of various features of these datasets can be found in (Jureczko and Madeyski 2010).

4.2 Evaluation metrics

To holistically assess the performance of our approach, we used EBEMs along with MLBEMs considering the realistic constraint of limited inspection effort. Typical MLBEMs like recall, precision, F-measure, AUC, etc. do not deliver much to practitioners in assessing the strength of a fault prediction model comprehensively. The model's cost-effectiveness and the amount of effort testers/developers have

to put in investigating faulty modules, are also important measures that need to be considered while evaluating any fault prediction model (Khatri and Singh 2021; Zhou et al. 2018) considering the constraint of limited inspection effort.

According to previous studies (Ostrand et al. 2005; Zhou et al. 2018), we considered that the inspection effort equal to 20% of TLOC (total lines of code) in the target data, is available at our disposal, taking lines of code (LOC) as the proxy for inspection effort.

4.2.1 MLBEMs

For holistic evaluation of our approach, we used three MLBEMs (G-measure, PF, and MCC). We specifically used G-measure and MCC to have a balanced performance for both fault-free and fault-prone classes. Thus, the accuracy of both classes is important (Ryu et al. 2017), therefore we considered G-measure and MCC. In particular, G-measure balances between recall and PF, whereas MCC considers all positive and negative predictions to calculate a composite score, which ranges between -1 and 1 . An MCC score of 1 represents the supreme performance, however, an MCC score of -1 shows the lowest performance. Further, we considered PF seeing its impact on the developer/tester's performance as high false positives lead to huge wastage of their effort and time and reduce their daily efficiency. Therefore, the lower the score, the superior the model.

These measures have been widely used in many other CPF studies (Chen et al. 2015; Zhou et al. 2018) and can be calculated as follows:

$$G - measure = \frac{2 * recall * (1 - PF)}{recall + (1 - PF)}$$

$$PF = \frac{FP}{FP + TN}$$

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Here, TP and TN denote the count of correct predictions of faulty and non-faulty classes respectively. On the other hand, FP and FN represent the count of false predictions of non-faulty and faulty classes respectively.

4.2.2 EBEMs

In reality, these MLBEMs just measure the correctness of the predictions but didn't provide enough information about the model's practical application. Since we have a limited inspection effort, therefore only a few faulty modules can go

Table 1 Dataset description

Short notation for dataset	Dataset name	Description	Count of modules	% of faulty modules
D1	ant 1.3	Open-source	125	16
D2	ant 1.4		178	22.47
D3	ant 1.5		293	10.92
D4	ant 1.6		351	26.21
D5	ant 1.7		745	22.28
D6	arc	Closed-source (Academic)	234	11.53
D7	berek		43	37.02
D8	camel 1.0	Open-source	339	3.83
D9	camel 1.2		608	35.52
D10	camel 1.4		872	16.62
D11	camel 1.6		965	19.48
D12	ckjm	Open-source	10	50
D13	elearn	Closed-source (Academic)	64	7.8
D14	forrest 0.7	Open-source	29	17.24
D15	forrest 0.8		32	6.25
D16	intercafe	Open-source	27	14.81
D17	ivy 1.1	Open-source	111	56.75
D18	ivy 1.4		241	6.64
D19	ivy 2.0		352	11.36
D20	jedit 3.2	Open-source	272	33.08
D21	jedit 4.0		306	24.5
D22	jedit 4.1		312	25.32
D23	jedit 4.2		367	13.08
D24	jedit 4.3		492	2.24
D25	kalkulator	Open-source	27	22.22
D26	log4j 1.0	Open-source	135	25.18
D27	log4j 1.1		109	33.94
D28	log4j 1.2		205	92.19
D29	lucene 2.0	Open-source	195	46.66
D30	lucene 2.2		247	58.29
D31	lucene 2.4		340	59.70
D32	nieruchomosci	Closed-source (Academic)	27	37.04
D33	pbeans 1.0	Open-source	26	76.92
D34	pbeans 2.0		51	19.60
D35	pdftranslator	Closed-source (Academic)	33	45.45
D36	poi 1.5	Open-source	237	59.49
D37	poi 2.0		314	11.78
D38	poi 2.5		385	64.42
D39	poi 3.0		442	63.57
D40	redaktor	Closed-source (Academic)	176	15.34
D41	serapion		45	20
D42	skarbonka		45	20
D43	sklebagd		20	60
D44	synapse 1.0	Open-source	157	10.19
D45	synapse 1.1		222	27.03
D46	synapse 1.2		256	33.59
D47	systemdata	Closed-source (Academic)	65	13.85
D48	szybkafucha		25	56.00
D49	termoproject		42	30.95

Table 1 (continued)

Short notation for dataset	Dataset name	Description	Count of modules	% of faulty modules
D50	tomcat	Open-source	858	8.97
D51	velocity 1.5	Open-source	214	66.36
D52	velocity 1.6		229	34.06
D53	workflow	Closed-source (Academic)	39	51.28
D54	wspomaganiepi		18	66.67
D55	xalan 2.4	Open-source	723	15.21
D56	xalan 2.5		803	48.19
D57	xalan 2.6		885	46.44
D58	xerces 1.0	Open-source	162	47.53
D59	xerces 1.2		440	16.14
D60	xerces 1.3		453	15.23
D61	xerces 1.4		588	74.32
D62	zuzel	Closed-source (Academic)	29	44.83

under inspection for locating fault positions and their corrections. To quantify the tester/developer's effort investment while inspecting these faulty modules and to measure the cost-effectiveness under the limited inspection effort of 20% of TLOC, we considered three EBEMs namely PIM@20%, Cost-effectiveness@20%, and IFA.

For better understanding, let's assume that there were 'P' modules in the target data, out of which 'D' modules were faulty. Further assume that 'p' modules were inspected under 20% of TLOC, out of which 'd' modules were faulty. Besides, consider that 'T' modules were investigated prior to noticing the first truly classified faulty module.

PIM@20% (Meyer et al. 2014) specifies the proportion of inspected modules under the 20% of TLOC and is computed as follows:

$$PIM@20\% = \frac{p}{P}$$

This measure accounts for the additional effort on testers/developers when they have to inspect more modules within the same inspection effort as they may need to switch between various files and require communication and coordination with people of different expertise. So, the lesser the score, the better the model.

Cost-effectiveness@20% (Huang et al. 2018) accounts for the actual performance of the model. It measures the proportion of faulty modules inspected under 20% of TLOC and can be calculated as follows:

$$Cost - effectiveness@20\% = \frac{d}{D}$$

The higher the score, the better the model.

IFA (initial false alarm) (Kochhar et al. 2016) measures the initial false predictions before encountering the first correctly classified faulty module. This measure is particularly

considered in the study as it strongly impacts the developer/tester's performance. A high IFA can shake their confidence in the model's capacity, causing fatigue and disappointment among them with a reduction in their work efficiency. Therefore, the lesser the score, the superior the performance of the model. It is computed as follows:

$$IFA = T$$

4.3 Research question and baseline selection

This section presents the research question and baselines selection.

To measure the true strength of the proposed model, we undertook the following research question:

Research Question Can the proposed approach surpass the Turhan-Filter (Turhan et al. 2009) and Cruz model (Cruz and Ochimizu 2009) comprehensively?

To answer this question, we performed several experiments taking 62 datasets, the result of which is presented in Sect. 5.1. We particularly selected these two state-of-the-art approaches namely Turhan-Filter (2009) and Cruz model (2009) for comparison as Herbold et al. (2018) confirmed these two approaches at the top in their large-scale comparative study amongst 24 existing approaches.

4.4 Experimental design

Algorithm 1 explains the experimental design of our proposed approach and Algorithm 2 explains the EBEMs computation process.

Algorithm 1 Experimental design of the proposed CFPF model

Input: All the available datasets

Output: Median performance of the proposed model with respect to MLBEMs and EBEMs

1. *AvailableData* = {D1, D2, D3, D4, …, D59, D60, D61, D62}
 2. For each *TargetData* in *AvailableData*
 - 2.1 *ReferenceData* = *AvailableData* – {*TargetData* + *TargetData_Versions*}
 - 2.2 Remove modules with Zero LOC from *TargetData* and *Reference Data*
 - 2.3 *Normalized_ReferenceData* = Normalize the *ReferenceData*
 - 2.4 *Normalized_TargetData* = Normalize the *TargetData*
 - 2.5 *Selected_Normalized_ReferenceData* = Apply KNN algorithm to select *Normalized_ReferenceData* similar to *Normalized_TargetData*
 - 2.6 *Proposed_Model* = Train the NB Model using *Selected_Normalized_ReferenceData*
 - 2.7 *Proposed_Model_Pred_Labels*, *Proposed_Model_Prob_Score* = Test the *Proposed_Model* on *Normalized_TargetData*
 - 2.8 Compute MLBEMs for *Proposed_Model*
 - 2.9 Calculate EBEMs for *Proposed_Model* using Algorithm 2
 3. Report the median performance over all datasets
-

First of all, we constructed the reference data for every target data by eliminating the target data and its related versions (if any) from the available pool of data (step 2.1). Next, we removed the modules having zero LOC from the target and the reference data as such modules did not contain any information for the fault (step 2.2). Next, we normalized the reference and the target data (steps 2.3 and 2.4) and then

selected the matching reference data for each target data by applying the KNN algorithm (step 2.5). After training data selection, the proposed model was constructed and validated and the various MLEBMs were calculated (steps 2.6 to 2.8). Next, the EBEMs were calculated by applying Algorithm 2 (step 2.9). In the end, the median results were reported over all the target projects (step 3).

Algorithm 2: EBEMs computation

Input: *Proposed_Model_Prob_Score*: Probability score generated by the proposed model
Proposed_Model_Pred_Labels: Predicted labels generated by the proposed model
Given_Inspection_Effort: 20 % of TLOC in *TargetData*

Output: *PIM@20%*, *Cost-effectiveness@20%*, *IFA*

- 1 Construct three vacant lists *L1_faulty*, *L2_Non-faulty* and *L3_Inspection*
- 2 For every module M^t in *TargetData*
 - 2.1 Put M^t into *L1_faulty*, if classified as faulty, otherwise put it in *L2_Non-faulty*
 - 2.2 Compute $risk_score = (Proposed_Model_Prob_Score / LOC) * avg_cc$
- 3 Arrange the modules in *L1_faulty* and *L2_Non-faulty* in decreasing order of their $risk_score$
- 4 Append *L2_Non-faulty* at the bottom of *L1_faulty*
- 5 Retrieve topmost modules from *L1_faulty* such that their TLOC should not exceed the given inspection effort and add them in *L3_Inspection*
- 6 Compute EBEMs using *L3_Inspection* as follows:

$$PIM@20\% = \frac{\text{Total modules in } L3_Inspection}{\text{Total modules in } TargetData}$$

$$Cost - effectiveness@20\% = \frac{\text{Faulty modules in } L3_Inspection}{\text{Faulty modules in } TargetData}$$

$$IFA = \text{Number of modules inspected prior observing the first truly classified faulty module in } L3_Inspection$$

Now, we explain the process of EBEMs computation as shown in Algorithm 2, which was designed following (Khatri and Singh 2022). First, three lists were constructed namely *L1_faulty*, *L2_Non-faulty*, and *L3_inspection*. All the faulty predicted modules were added in *L1_faulty* and all non-faulty predicted modules were added in *L2_Non-faulty*. Next, the $risk_score$ was calculated for every target data module according to step 2.2. This $risk_score$ took three inputs namely the probability score produced by the model, the average cyclomatic complexity (avg_cc), and the LOC. Both the probability score and the cyclomatic complexity represent the proxies for the module's fault proneness and the third input i.e., LOC represent the proxy for the inspection effort. The higher the probability score and the cyclomatic complexity, the more the fault-proneness. The more the LOC, the higher the inspection effort. Therefore, we designed our $risk_score$ such that the modules with higher fault-proneness, having low inspection effort must be investigated first.

Keeping this rationale in mind, the two lists were then arranged in descending order of their $risk_score$. After that, the list *L2_Non-faulty* was appended at the bottom of the list *L1_faulty*. Next, we extracted some topmost modules from *L1_faulty*, with their TLOC equal to the given inspection effort, and added them to the list *L3_Inspection*. Based on this list, the EBEMs were then calculated according to step 6.

5 Results and discussions

This section discusses the results, its analysis, and the ablation study to examine the contribution of each phase in the overall performance of the proposed CPF model.

5.1 Results

Table 2 depicts the result of the comparison of our proposed model with the Turhan-Filter and Cruz model in

Table 2 Comparison of our proposed approach with Turhan-Filter and Cruz model in terms of MLBEMs and EBEMs

Dataset	Proposed approach						Turhan-Filter						Cruz					
	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA
D1	0.757	0.411	0.346	0.169	0.350	7	0.486	0.226	0.673	0.347	0.300	26	0.627	0.284	0.519	0.250	0.350	14
D2	0.562	0.122	0.358	0.169	0.250	1	0.458	0.111	0.679	0.339	0.400	8	0.569	0.117	0.460	0.209	0.275	3
D3	0.730	0.302	0.315	0.182	0.250	17	0.548	0.194	0.608	0.332	0.219	53	0.691	0.253	0.381	0.219	0.219	23
D4	0.761	0.470	0.260	0.191	0.250	0	0.588	0.322	0.566	0.354	0.228	3	0.707	0.396	0.388	0.243	0.217	0
D5	0.708	0.355	0.295	0.199	0.217	6	0.594	0.308	0.561	0.327	0.301	7	0.700	0.371	0.405	0.255	0.271	4
D6	0.580	0.226	0.167	0.116	0.148	3	0.595	0.132	0.359	0.196	0.148	4	0.585	0.251	0.146	0.102	0.185	3
D7	0.814	0.615	0.185	0.256	0.375	5	0.814	0.648	0.111	0.233	0.438	3	0.726	0.524	0.407	0.326	0.313	9
D8	0.665	0.144	0.276	0.131	0.231	0	0.539	0.052	0.559	0.269	0.231	27	0.518	0.083	0.208	0.107	0.231	8
D9	0.478	0.100	0.257	0.142	0.171	0	0.535	0.069	0.488	0.256	0.259	4	0.419	0.105	0.192	0.121	0.139	0
D10	0.593	0.218	0.222	0.134	0.208	0	0.644	0.233	0.431	0.234	0.278	12	0.502	0.171	0.176	0.116	0.174	0
D11	0.523	0.160	0.219	0.130	0.197	0	0.570	0.118	0.474	0.253	0.261	0	0.485	0.128	0.214	0.129	0.186	0
D12	0.000	-0.333	1.000	0.400	0.000	4	0.320	0.000	0.800	0.300	0.000	3	0.320	0.000	0.200	0.100	0.000	1
D13	0.846	0.516	0.102	0.094	0.200	0	0.487	0.189	0.678	0.313	0.000	20	0.566	0.406	0.034	0.047	0.200	0
D14	0.345	0.208	0.792	0.379	0.400	6	0.646	0.258	0.458	0.138	0.200	2	0.632	0.208	0.333	0.103	0.000	3
D15	0.696	0.258	0.467	0.188	0.500	5	0.667	0.243	0.500	0.188	0.500	4	0.531	0.033	0.433	0.125	0.000	4
D16	0.672	0.256	0.391	0.222	0.000	6	0.647	0.346	0.522	0.259	0.250	2	0.606	0.320	0.565	0.222	0.000	6
D17	0.593	0.309	0.167	0.153	0.190	0	0.601	0.269	0.229	0.189	0.222	1	0.619	0.306	0.208	0.198	0.238	0
D18	0.648	0.186	0.236	0.141	0.375	10	0.661	0.175	0.298	0.212	0.313	32	0.675	0.197	0.267	0.158	0.188	15
D19	0.710	0.310	0.218	0.134	0.175	10	0.691	0.257	0.375	0.264	0.200	36	0.724	0.311	0.250	0.156	0.150	14
D20	0.746	0.469	0.253	0.250	0.456	0	0.629	0.326	0.495	0.353	0.478	6	0.704	0.413	0.385	0.316	0.467	0
D21	0.695	0.343	0.303	0.239	0.347	8	0.656	0.313	0.450	0.304	0.427	12	0.665	0.312	0.424	0.294	0.333	2
D22	0.712	0.385	0.356	0.260	0.380	0	0.596	0.259	0.532	0.333	0.342	16	0.638	0.316	0.489	0.321	0.329	0
D23	0.719	0.351	0.417	0.262	0.417	0	0.530	0.248	0.636	0.387	0.354	0	0.569	0.255	0.596	0.338	0.313	0
D24	0.592	0.056	0.447	0.220	0.000	108	0.478	0.006	0.617	0.337	0.000	166	0.521	0.023	0.559	0.283	0.000	139
D25	0.279	0.028	0.143	0.074	0.167	0	0.392	-0.158	0.524	0.074	0.167	0	0.279	0.028	0.143	0.074	0.167	0
D26	0.664	0.442	0.111	0.105	0.265	1	0.499	0.248	0.657	0.331	0.294	0	0.629	0.503	0.051	0.083	0.265	0
D27	0.754	0.644	0.042	0.110	0.243	1	0.546	0.323	0.611	0.294	0.135	7	0.717	0.626	0.028	0.101	0.243	0
D28	0.400	0.083	0.125	0.137	0.143	0	0.614	0.124	0.375	0.263	0.280	0	0.386	0.115	0.063	0.117	0.122	0
D29	0.681	0.370	0.269	0.221	0.253	1	0.662	0.328	0.375	0.262	0.286	1	0.659	0.338	0.260	0.185	0.198	3
D30	0.596	0.222	0.291	0.190	0.215	0	0.628	0.255	0.388	0.291	0.347	0	0.604	0.256	0.252	0.174	0.215	0
D31	0.605	0.299	0.190	0.179	0.227	0	0.637	0.270	0.350	0.271	0.310	0	0.633	0.313	0.219	0.197	0.241	1
D32	0.381	0.320	0.765	0.259	0.200	2	0.692	0.542	0.471	0.222	0.200	1	0.182	0.256	0.000	0.074	0.200	0
D33	0.000	-0.208	1.000	0.600	0.579	2	0.400	0.096	0.167	0.160	0.158	0	0.273	0.208	0.000	0.040	0.053	0
D34	0.459	0.404	0.026	0.041	0.100	0	0.733	0.403	0.231	0.245	0.500	0	0.558	0.372	0.077	0.082	0.200	0

Table 2 (continued)

Dataset	Proposed approach						Turhan-Filter						Cruz					
	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA	G-measure	MCC	PF	PIM@20%	Cost-effectiveness@20%	IFA
D35	0.696	0.620	0.000	0.091	0.200	0	0.875	0.784	0.222	0.394	0.600	0	0.421	0.407	0.000	0.061	0.133	0
D36	0.664	0.332	0.375	0.257	0.284	0	0.453	0.270	0.698	0.392	0.340	1	0.616	0.342	0.510	0.321	0.319	0
D37	0.617	0.156	0.359	0.176	0.189	2	0.303	0.086	0.819	0.454	0.432	3	0.469	0.098	0.670	0.348	0.297	1
D38	0.656	0.314	0.265	0.245	0.294	2	0.513	0.378	0.647	0.471	0.496	9	0.650	0.373	0.463	0.357	0.415	7
D39	0.775	0.533	0.188	0.283	0.395	1	0.541	0.391	0.619	0.485	0.484	0	0.775	0.570	0.294	0.347	0.448	2
D40	0.395	0.092	0.743	0.297	0.444	19	0.228	0.149	0.872	0.383	0.481	4	0.498	0.133	0.642	0.240	0.259	13
D41	0.741	0.452	0.167	0.089	0.111	3	0.648	0.267	0.444	0.289	0.111	2	0.700	0.552	0.056	0.089	0.222	2
D42	0.780	0.511	0.361	0.200	0.556	1	0.640	0.315	0.500	0.267	0.333	2	0.617	0.209	0.306	0.156	0.222	1
D43	0.536	0.171	0.250	0.100	0.083	1	0.600	0.257	0.500	0.200	0.167	0	0.647	0.471	0.500	0.200	0.083	3
D44	0.749	0.321	0.305	0.134	0.250	4	0.544	0.203	0.617	0.217	0.063	9	0.560	0.178	0.589	0.191	0.188	11
D45	0.615	0.264	0.241	0.135	0.150	7	0.602	0.207	0.309	0.180	0.217	6	0.637	0.258	0.426	0.180	0.133	2
D46	0.655	0.354	0.207	0.141	0.174	3	0.695	0.370	0.320	0.192	0.244	0	0.657	0.307	0.396	0.200	0.140	0
D47	0.706	0.312	0.250	0.185	0.222	1	0.625	0.199	0.286	0.169	0.111	7	0.553	0.134	0.268	0.200	0.222	2
D48	0.465	0.007	0.636	0.360	0.286	0	0.239	-0.161	0.273	0.440	0.429	0	0.353	0.327	0.000	0.040	0.071	0
D49	0.609	0.403	0.103	0.143	0.231	0	0.739	0.464	0.345	0.262	0.231	3	0.143	0.233	0.000	0.000	0.000	0
D50	0.033	0.039	0.983	0.630	0.260	121	0.551	0.167	0.181	0.126	0.247	38	0.746	0.330	0.203	0.125	0.312	5
D51	0.378	0.166	0.100	0.151	0.197	0	0.657	0.315	0.400	0.358	0.401	0	0.396	0.180	0.100	0.142	0.183	1
D52	0.467	0.212	0.141	0.163	0.231	1	0.562	0.209	0.557	0.383	0.436	0	0.469	0.232	0.128	0.145	0.205	1
D53	0.612	0.230	0.421	0.410	0.450	1	0.542	0.269	0.158	0.128	0.150	0	0.531	0.205	0.211	0.154	0.100	1
D54	0.741	0.471	0.167	0.222	0.333	0	0.800	0.756	0.333	0.222	0.250	1	0.909	0.791	0.000	0.222	0.333	0
D55	0.707	0.309	0.349	0.235	0.291	7	0.658	0.285	0.469	0.297	0.309	29	0.637	0.253	0.485	0.310	0.282	25
D56	0.502	0.106	0.287	0.222	0.214	2	0.573	0.145	0.426	0.332	0.336	0	0.546	0.119	0.355	0.271	0.256	2
D57	0.541	0.238	0.189	0.222	0.277	0	0.558	0.115	0.447	0.367	0.360	0	0.563	0.177	0.298	0.283	0.304	0
D58	0.454	0.147	0.188	0.177	0.178	2	0.470	-0.027	0.424	0.304	0.233	11	0.421	0.235	0.094	0.108	0.137	1
D59	0.377	0.155	0.100	0.061	0.090	0	0.521	0.086	0.307	0.217	0.269	2	0.396	0.166	0.102	0.063	0.104	1
D60	0.440	0.230	0.086	0.059	0.116	0	0.751	0.393	0.237	0.206	0.464	1	0.421	0.188	0.102	0.066	0.116	2
D61	0.289	0.202	0.014	0.066	0.089	0	0.598	0.376	0.027	0.216	0.286	0	0.254	0.183	0.014	0.056	0.073	0
D62	0.819	0.651	0.125	0.172	0.231	1	0.788	0.617	0.313	0.310	0.385	1	0.860	0.721	0.125	0.207	0.308	1

Table 3 Median performance in terms of MLBEMs and EBEMs

Evaluation metrics	Proposed model	Turhan-Filter	Cruz
G-measure ↑	0.615	0.595	0.577
PF ↓	0.251	0.454	0.251
MCC↑	0.282	0.256	0.254
PIM@20% ↓	0.178	0.271	0.167
Cost-effectiveness@20% ↑	0.231	0.283	0.202
IFA ↓	1	2	1

‘↑’ signifies ‘the greater the better’; ‘↓’ indicates ‘the lesser the better’

Table 4 Statistical test results

Evaluation metrics		Turhan-Filter	Cruz
G-measure	p-value for our proposed model vs	0.68	0.015 (+)
PF		0.000 (+)	0.344
MCC		0.078	0.294
PIM@20%		0.000 (+)	0.476
Cost-effectiveness@20%		0.009 (–)	0.063
IFA		0.006 (+)	0.34

‘+’ signifies ‘better performance of the proposed model’; ‘–’ indicates ‘the lower performance of the proposed model’

terms of G-measure, PF, MCC, PIM@20%, Cost-effectiveness@20%, and IFA scores over 62 datasets. Further ahead, Table 3 summarizes the results by taking their medians for all compared models. We use median results to eradicate the effect of outliers as they can mislead the results.

While comparing to *Turhan-Filter*, our model dominated in all MLBEMs with an improvement of 3.36%, 80.87%, and 10.15%, concerning median G-measure, median PF, and median MCC scores respectively. Further, concerning EBEMs, our proposed model showed its superiority with respect to PIM@20% and IFA with an improvement of 52.24% and 100% in their respective median scores. However, the performance with respect to Cost-effectiveness@20% was a little low.

Next, while comparing with the *Cruz model*, our model showed its excellence with respect to G-measure and MCC with similar performance concerning PF. In particular, our proposed model achieved an improvement of approximately 7% and 11% in median G-measure and median MCC scores respectively. Further, with respect to median cost-effectiveness@20%, it showed an improvement of 14.35% with similar performance in other metrics namely PIM@20% and IFA.

We further applied a statistical test to compare our proposed approach with *Turhan-Filter* and *Cruz model* statistically. In particular, we applied the Wilcoxon-Signed Rank test (Demšar 2006) at a 5% significance level for statistical comparison. Being a non-parametric test, it compares two related samples to verify their belongingness to the same population. The resulting p-value score of more than or equal to 0.05 reveals that the two compared samples do not differ statistically. However, the p-value score of less than 0.05 reveals that the two compared samples differ statistically. Table 4 depicts the statistical test results.

The results of the statistical test also revealed the supremacy of our proposed approach over the *Turhan-Filter* concerning PF, PIM@20%, and IFA as the p-values were less than 0.05. The two compared approaches performed comparably to each other with respect to G-measure and MCC (as p-values > 0.05). However, with respect to Cost-effectiveness@20%, the performance was a little low as compared to *Turhan-Filter*.

Further, these results also confirmed the excellence of our proposed approach over the *Cruz model* particularly with regard to G-measure as the p-value was less than 0.05 with comparable performance with respect to other metrics. However, we observed an improvement of approximately 11% and 14.35% in terms of median MCC and median Cost-effectiveness@20% scores respectively over the *Cruz model*.

5.2 Discussions

To analyse our results concretely, we make use of boxplots to pictorially represent the distribution of various evaluation metrics obtained from our proposed model, the *Turhan-Filter* and *Cruz model* over all the 62 datasets as shown in Fig. 3. The lower black horizontal line of each box shows the first quartile, the center black line shows the median (i.e., 2nd quartile) and the upper black horizontal line of the box shows the 3rd quartile. The lowermost black horizontal line and the uppermost black horizontal line depict the minimum and maximum values of the distribution.

From Fig. 3a, we conclude the superiority of our model in comparison to *Cruz model* with similar performance to *Turhan-Filter* in terms of G-measure. Next, from Fig. 3b, we observed the similar performance of all three models in terms of MCC. Further, Fig. 3c revealed the outstanding performance of our proposed model and the *Cruz model* in comparison to *Turhan-Filter* concerning PF. However, with respect to PF, our proposed model and the *Cruz model* performed comparable to each other. Similarly, with respect to PIM@20%, the *Cruz model* and our proposed model performed comparably to each other as evident in Fig. 3d.

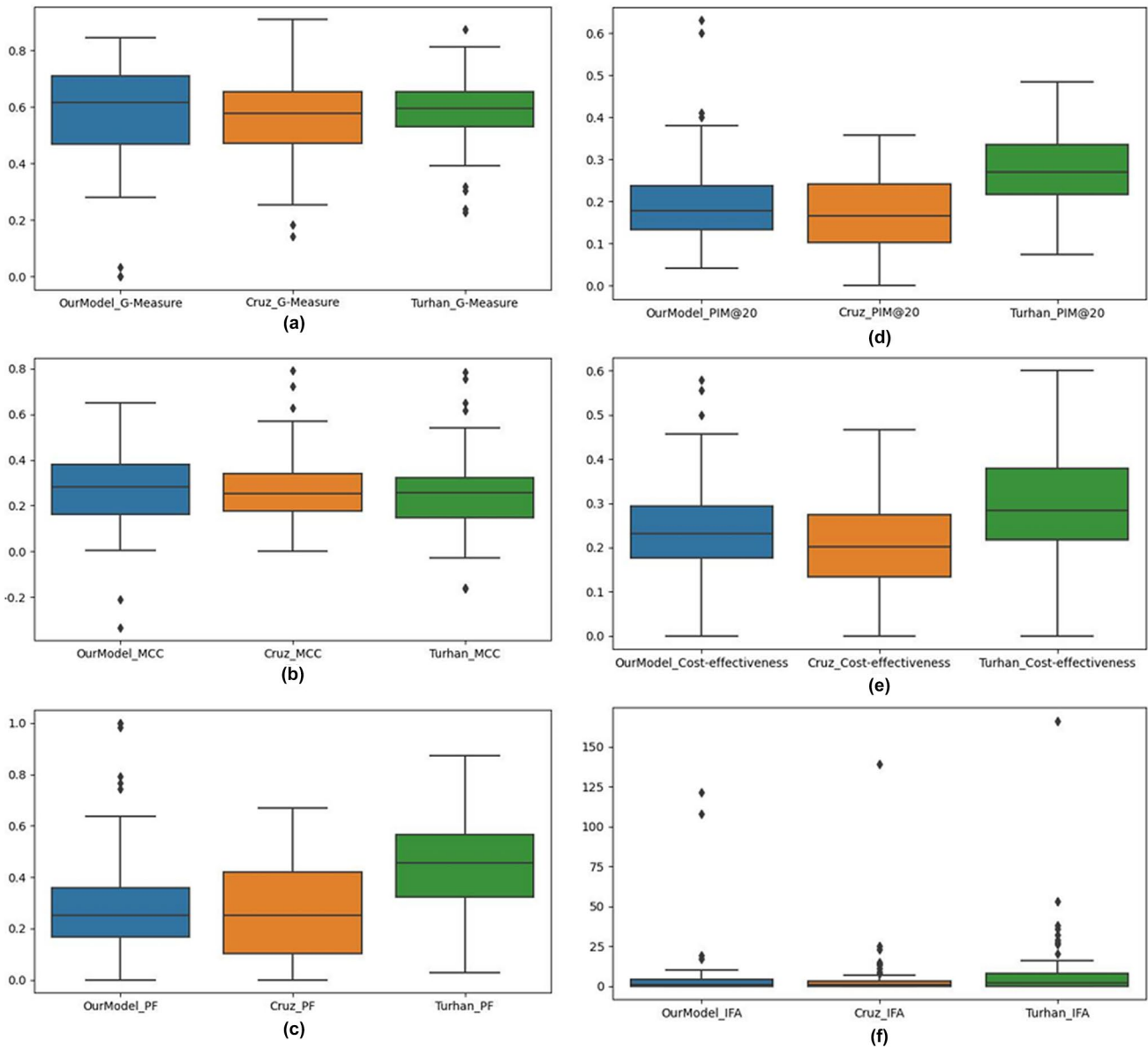


Fig. 3 Boxplots of MLBEMs and EBEMs

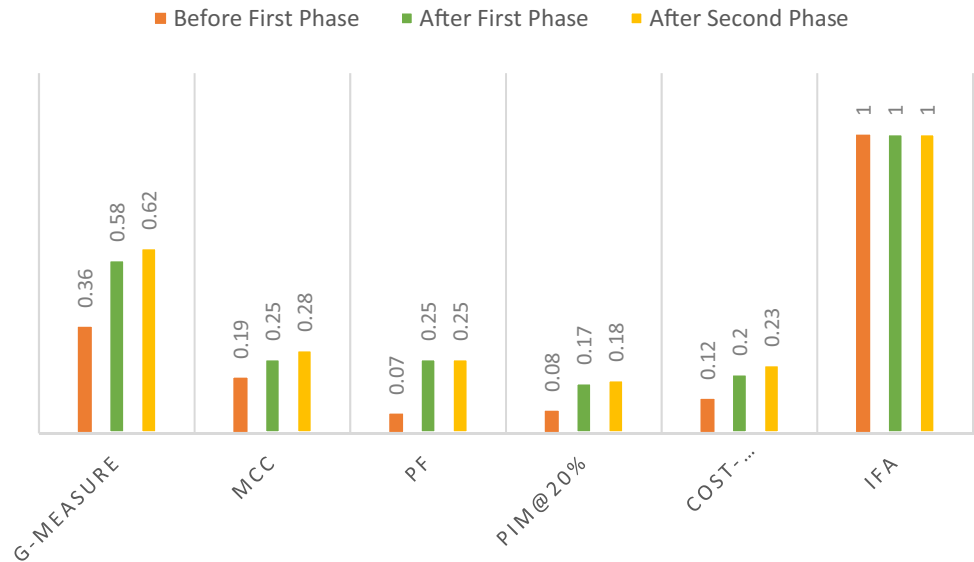
However, in comparison to Turhan-Filter, both performed much better. Further as visible in Fig. 3e, our model performed better than the Cruz model, but Turhan-Filter outperformed both in terms of Cost-effectiveness@20%. In particular, Turhan-Filter achieved the best performance with respect to Cost-effectiveness@20% at the cost of PIM@20%. Because the proportion of inspected modules happened to be very large in the case of Turhan-Filter, therefore it succeeded in inspecting more faulty modules, putting a huge additional effort on developers/testers. Further, with respect to IFA, Turhan-Filter performed the worst as evident in Fig. 3f, whereas the Cruz model and our model performed comparably to each other. Thus, combining all evaluation

metrics, we claimed the dominance of our proposed model in comparison to Turhan-Filter and the Cruz model.

5.3 Ablation study

To witness the impact of each and every phase on the model’s performance, we performed an ablation study. In particular, first we measured the model’s performance before the first phase of normalization, i.e., taking the data as it is, i.e., without applying any normalization technique. After that, we captured the model’s performance after applying the first phase of normalization to particularly investigate the impact of this phase on the model’s performance. Next, we

Fig. 4 Impact of different phases on the proposed model's performance



measured the performance after applying our second phase of training data selection in sequence after the first phase to examine the impact of this phase on the model's performance. Figure 4 shows the performance of the proposed CPF model in terms of median G-measure, PF, MCC, PIM@20%, Cost-effectiveness@20% and IFA scores, calculated over 62 CPF experiments, before the normalization phase, after the first phase of normalization and second phase of training data selection respectively.

It is evident from the above figure that the normalization phase brought substantial improvement in the proposed model's prediction performance. In other words, in terms of MLBEMs, the normalization phase improved the median G-measure and MCC scores by approximately 61% and 32% respectively with a slight surge in the PF score. Since both faulty and non-faulty classes are important, therefore giving equal weightage to the accuracy of both classes, the normalization phase achieved improvement in the overall performance in terms of MLBEMs by showcasing better performance in two compound evaluation metrics i.e., G-measure and MCC. Further, in terms of EBEMs, the normalization phase brought improvement in Cost-effectiveness@20% with a slight surge in PIM@20%. However, the performance in terms of IFA remained the same. In other words, the performance in terms of EBEMs more or less remained the same after the normalization phase. But combining all MLBEMs and EBEMs, the performance of the proposed CPF model significantly improved after the application of the first phase of normalization.

Further ahead, the application of second phase of training data selection after the first phase further improved the proposed model's performance by achieving an improvement of approximately 7% and 12% in median G-measure and MCC scores respectively with comparable performance in terms

of median PF score. Furthermore, in terms of EBEMs, it witnessed an improvement of approximately 15% in median Cost-effectiveness@20% score with comparable performance in terms of median PIM@20% and IFA scores.

Thus, the above ablation study showcased the impact of normalization phase and training data selection phase on the proposed model's performance. Both phases improved the overall performance (i.e., combining MLBEMs and EBEMs) and thus contributed significantly to justify the effectiveness of the proposed CPF model.

6 Threats to validity

Some of the significant issues that can pose threats to our proposed work are mentioned as follows:

6.1 Internal validity

To examine the strength of our proposed model, we have used a large corpus of 62 datasets, wherein each dataset contains object-oriented, complexity-based, and size-based software metrics as the potential fault predictors. These software metrics have already proven their importance as potential fault predictors in many SFP studies (Basili et al. 1996; Jureczko and Spinellis 2010; Subramanyam and Krishnan 2003). However, we cannot assure the same performance of our model on projects containing other software metrics.

6.2 External validity

This threat relates to validating the effectiveness of the proposed work on other software projects from different domains, developed under diverse operating procedures

and principles. However, to minimize this threat to some extent, we have validated the performance of our approach on 62 datasets, consisting of a good mix of both open and closed-source projects belonging to different domains.

7 Conclusion

This study aims at building quality software with lesser maintenance cost using an improved cross-project fault prediction (CPFP) model. The proposed model consisted of three phases. The first phase normalized the reference and the target data, the second phase selected the relevant training data by applying the KNN algorithm and the third phase finally constructed the CPFP model. To assess the strength of the proposed model comprehensively, the effort-based evaluation metrics (EBEMs) along with MLBEMs (machine learning-based evaluation metrics) were considered, taking the realistic constraint of limited inspection effort. Based on the large-scale empirical investigation on 62 datasets, the excellence of the proposed model over state-of-the-art models namely the Turhan-Filter and the Cruz model was concluded. In the future, we will work on further improving its performance by building hybrid models combining normalization, instance selection and feature selection with its validation on more datasets.

Funding No funds, grants, or other support was received.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Human and/or animals participants This study doesn't involve any human/animal participants.

Informed consent This study does not involve any human/animal participants, as a result no consent is needed.

References

- Arar ÖF, Ayan K (2015) Software defect prediction using cost-sensitive neural network. *Appl Soft Comput* 33:263–277. <https://doi.org/10.1016/j.asoc.2015.04.045>
- Basili VR, Briand LC, Melo WL (1996) A validation of object-oriented design metrics as quality indicators. *IEEE Trans Software Eng* 22(10):751–761. <https://doi.org/10.1109/32.544352>
- Bisi M, Goyal NK (2016) An ANN-PSO-based model to predict fault-prone modules in software. *Int J Reliab Saf* 10(3):243–264. <https://doi.org/10.1504/IJRS.2016.081611>
- Bowes D, Hall T, Petrić J (2018) Software defect prediction: do different classifiers find the same defects? *Softw Qual J* 26(2):525–552. <https://doi.org/10.1007/s11219-016-9353-3>
- Briand LC, Melo WL, Wüst J (2002) Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans Softw Eng* 28(7):706–720. <https://doi.org/10.1109/TSE.2002.1019484>
- Canfora G, Lucia AD, Penta MD, Oliveto R, Panichella A, Panichella S (2015) Defect prediction as a multiobjective optimization problem. *Softw Test Verif Reliab* 25(4):426–459. <https://doi.org/10.1002/STVR.1570>
- Chen L, Fang B, Shang Z, Tang Y (2015) Negative samples reduction in cross-company software defects prediction. *Inf Softw Technol* 62(1):67–77. <https://doi.org/10.1016/j.infsof.2015.01.014>
- Cover TM, Hart PE (1967) Nearest neighbor pattern classification. *IEEE Trans Inf Theory* 13(1):21–27. <https://doi.org/10.1109/TIT.1967.1053964>
- Cruz AEC, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects. In: 2009 3rd International symposium on empirical software engineering and measurement, pp 460–463. <https://doi.org/10.1109/ESEM.2009.5316002>
- D'Ambros M, Lanza M, Robbes R (2012) Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empir Softw Eng* 17(4–5):531–577. <https://doi.org/10.1007/s10664-011-9173-9>
- Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res* 7:1–30
- He P, Li B, Liu X, Chen J, Ma Y (2015) An empirical study on software defect prediction with a simplified metric set. *Inf Softw Technol* 59:170–190. <https://doi.org/10.1016/j.infsof.2014.11.006>
- Herbold S, Trautsch A, Grabowski J (2018) A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans Softw Eng* 44(9):811–833. <https://doi.org/10.1109/TSE.2017.2724538>
- Herbold S (2013) Training data selection for cross-project defect prediction. In: ACM international conference proceeding series, Part F1288, pp 1–10. <https://doi.org/10.1145/2499393.2499397>
- Hosseini S, Turhan B, Gunarathna D (2019) A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Trans Softw Eng* 45(2):111–147. <https://doi.org/10.1109/TSE.2017.2770124>
- Huang Q, Xia X, Lo D (2018) Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empir Softw Eng* 24(5):2823–2862. <https://doi.org/10.1007/s10664-018-9661-2>
- Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: ACM international conference proceeding series, pp 1–10. <https://doi.org/10.1145/1868328.1868342>
- Jureczko M, Spinellis D (2010) Using object-oriented design metrics to predict software defects. In: Models and methods of system dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, pp 69–81. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.226.2285>
- Kassab M, DeFranco JF, Laplante PA (2017) Software testing: the state of the practice. *IEEE Softw* 34(5):46–52. <https://doi.org/10.1109/MS.2017.3571582>
- Kawata K, Amasaki S, Yokogawa T (2015) Improving relevancy filter methods for cross-project defect prediction. In: Proceedings—3rd international conference on applied computing and information technology and 2nd international conference on computational science and intelligence, ACIT-CSI 2015, pp 2–7. <https://doi.org/10.1109/ACIT-CSI.2015.104>
- Khatri Y, Singh SK (2021) Cross project defect prediction: a comprehensive survey with its SWOT analysis. *Innov Syst Softw Eng*. <https://doi.org/10.1007/s11334-020-00380-5>

- Khatri Y, Singh SK (2022) Towards building a pragmatic cross-project defect prediction model combining non-effort based and effort-based performance measures for a balanced evaluation. *Inf Softw Technol* 150:106980. <https://doi.org/10.1016/J.INFSOF.2022.106980>
- Kochhar PS, Xia X, Lo D, Li S (2016) Practitioners' expectations on automated fault localization. In: Proceedings of the 25th international symposium on software testing and analysis, pp 165–176. <https://doi.org/10.1145/2931037.2931051>
- Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Trans Softw Eng* 34(4):485–496. <https://doi.org/10.1109/TSE.2008.35>
- Liu Y, Khoshgoftaar TM, Seliya N (2010) Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Trans Softw Eng* 36(6):852–864. <https://doi.org/10.1109/TSE.2010.51>
- Lu H, Cukic B, Culp M (2012) Software defect prediction using semi-supervised learning with dimension reduction. In: 2012 27th IEEE/ACM international conference on automated software engineering, ASE 2012 —Proceedings, pp 314–317. <https://doi.org/10.1145/2351676.2351734>
- Ma Y, Luo G, Zeng X, Chen A (2012) Transfer learning for cross-company software defect prediction. *Inf Softw Technol* 54(3):248–256. <https://doi.org/10.1016/j.infsof.2011.09.007>
- Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng* 33(1):2–13. <https://doi.org/10.1109/TSE.2007.256941>
- Meyer AN, Fritz T, Murphy GC, Zimmermann T (2014) Software developers' perceptions of productivity. In: Proceedings of the ACM SIGSOFT symposium on the foundations of software engineering, 16–21 November, pp 19–29. <https://doi.org/10.1145/2635868.2635892>
- Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: Proceedings—international conference on software engineering, pp 382–391. <https://doi.org/10.1109/ICSE.2013.6606584>
- Ostrand TJ, Weyuker EJ, Bell RM (2005) Predicting the location and number of faults in large software systems. *IEEE Trans Softw Eng* 31(4):340–355. <https://doi.org/10.1109/TSE.2005.49>
- Pan SJ, Tsang IW, Kwok JT, Yang Q (2011) Domain adaptation via transfer component analysis. *IEEE Trans Neural Netw* 22(2):199–210. <https://doi.org/10.1109/TNN.2010.2091281>
- Pelayo L, Dick S (2007) Applying novel resampling strategies to software defect prediction. In: Annual conference of the north American fuzzy information processing society—NAFIPS, pp 69–72. <https://doi.org/10.1109/NAFIPS.2007.383813>
- Peng L, Yang B, Chen Y, Abraham A (2009) Data gravitation based classification. *Inf Sci* 179(6):809–819. <https://doi.org/10.1016/j.ins.2008.11.007>
- Ryu D, Jang JI, Baik J (2017) A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw Qual J* 25(1):235–272. <https://doi.org/10.1007/s11219-015-9287-1>
- Subramanyam R, Krishnan MS (2003) Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. *IEEE Trans Softw Eng* 29(4):297–310. <https://doi.org/10.1109/TSE.2003.1191795>
- Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. *Empir Softw Eng* 14:540–578. <https://doi.org/10.1007/s10664-008-9103-7>
- Wang T, Zhang Z, Jing X, Zhang L (2015) Multiple kernel ensemble learning for software defect prediction. *Autom Softw Eng* 23(4):569–590. <https://doi.org/10.1007/S10515-015-0179-1>
- Watanabe S, Kaiya H, Kajiri K (2008) Adapting a fault prediction model to allow inter language reuse. In: Proceedings—international conference on software engineering, pp 19–24. <https://doi.org/10.1145/1370788.1370794>
- Zhou Y, Yang Y, Lu H, Chen L, Li Y, Zhao Y, Qian J, Xu B (2018) How far we have progressed in the journey? An examination of cross-project defect prediction. *ACM Trans Softw Eng Methodol* 27(1):1–51. <https://doi.org/10.1145/3183339>
- Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: ESEC-FSE'09—Proceedings of the joint 12th European software engineering conference and 17th ACM SIGSOFT symposium on the foundations of software engineering, pp 91–100. <https://doi.org/10.1145/1595696.1595713>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.