An Empirical Evaluation of Defect Prediction Models Using Project-Specific Measures

Umamaheswara Sharma B[1,*,†], Ravichandra Sadam[2,†]

[1]Research Scholar, Department of Computer Science and Engineering, National Institute of Technology, Warangal, Telangana, India
[2]Associate Professor, Department of Computer Science and Engineering, National Institute of Technology, Warangal, Telangana, India

Abstract

Due to the advantages of economizing the testing resources such as cost, time, and consequently the manpower on the developing software project, research on software defect prediction (SDP) has gained traction in academia. Though many works in the literature discuss constraints that are limiting the final prediction performance, finding the essential benefits in terms of the project objectives such as cost, service time, and failure is rarely explored. On the basis of these project objectives, the gap of finding the best performing SDP model is still present in the literature. In this regard, in this work, a detailed empirical analysis of With-in Project Defect Prediction (WPDP), Cross-Project Defect Prediction (CPDP), and *mixed*-Cross-Project Defect Prediction (M-CPDP) models is provided using the project-specific performance measures such as percent of perfect cleans (PPC), percent of non-perfect cleans (PNPC), false omission rate (FOR), and its additional derived performance measures, which are proposed by Sharma et al. in [1]. The empirical analysis is provided on 14 publicly available datasets collected from the PROMISE repository using the baselines such as support vector machines (SVM), decision trees (DT), and k-nearest neighbours (k-NN) classifiers. From the empirical results, we observe that the M-CPDP model is significantly better at providing maximum savings in the allocated budget, minimum service time, and minimum failure incidents on the majority of the target projects.

Keywords

With-in Project Defect Prediction, Mixed Cross-Project Defect Prediction, Cross-Project Defect Prediction, project-specific Performance Measures, Prediction Quality Assessment

1. Introduction

Software defect prediction (SDP) models reduce the work load on the tester by providing the status of defect-proneness of the newly developed software module in a short time [2, 3, 4, 5, 6, 7, 8, 9, 10]. Hence, it reduces the total cost, time, and manpower that are spent on the target project [11, 1]. Because of these advantages, there are the works in the literature that address various constraints in building prediction models [2, 7, 12, 13, 14, 15, 16, 17, 18, 19, 20].

The literature on SDP classifies works on its major categories such as within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). WPDP models use available local data from the same software to train the prediction model, whereas CPDP models use defects data collected from multiple projects to train the prediction model[21]. The CPDP models are further classified into many types, such as *mixed* CPDP (M-CPDP), mixed project defect prediction (MPDP), and *pair-wise*-CPDP [21]. In *mixed* CPDP, old versions of the same project along with the data of the other source projects are used to train the prediction model. Hence, the M-CPDP models are considered as a mixture of with-in and cross-project defect prediction contexts. In MPDP, the prediction is trained using the data from the target project along with the old versions of the same project and the data of the other source projects. Whereas in the *pair-wise* CPDP, the prediction models will be built using each project's data. Then, the mean or median of the performances of these pair-wise predictions is then used to calculate the final performance of the *pair-wise* CPDP model. Among all the variants of SDP, most literature discusses works on WPDP, CPDP, and M-CPDP [1, 10, 21] models.

The common objective of developing SDP models is to reduce the project cost that is being spent on the testing team, reduce the work load on the tester, and minimise the risk of observing the failures [1, 22]. Owing to these objectives, recently, a work by Sharma et al. in [1] discusses various novel project-specific performance measures such as percent of perfect cleans (PPC), percent of non-perfect cleans (PNPC), false-omission rate (FOR), percent of saved budget (PSB), and percent of remaining edits (PRE), to capture real-benefits from the prediction model. These performance measures are essential in understanding the main objectives of the prediction model but are also used to evaluate the developed prediction model.

Further extending the work of Sharma et al. [1], in

QuASoQ 2022: 10th International Workshop on Quantitative Approaches to Software Quality, December 06, 2022, virtual
*Corresponding author
†These authors contributed equally in this research.
uma.phd@student.nitw.ac.in (U. S. B); ravic@nitw.ac.in (R. Sadam)
0000-0003-1676-3347 (U. S. B)

this work, we provide an empirical analysis of the performances of the widely used SDP variants such as WPDP, CPDP, and M-CPDP models using the project-specific performance measures. For the empirical analysis, we have used benchmark classification approaches such as support vector machines (SVM), decision trees (DT), and k-nearest neighbours (k-NN), for each problem context. From the empirical analysis, we observe that the M-CPDP model is performing significantly better over the others in terms of the project-specific attributes.

1.1. Contributions

Capturing the project benefits from the SDP model and obtaining the best performing model is essential for any research practitioner. Hence, this work is mainly targeted at providing an empirical analysis of the SDP models such as WPDP, CPDP, and M-CPDP with regard to the project-specific performance measures such as percent of perfect cleans (PPC), percent of non-perfect cleans (PNPC), false-omission rate (FOR), percent of saved budget (PSB), and percent of remaining edits (PRE), to capture real-benefits from the best prediction model. Since interpreting the obtained results of the SDP model in terms of the project objectives is a necessary task, any research practitioner should have to evaluate their models using the project-specific measures as in [1]. To the best of our knowledge, conducting an empirical analysis between the prediction models such as WPDP, CPDP, and M-CPDP in order to know the best performing model in terms of the SDP objectives is new to this field of research.

Paper Organisation: The rest of the paper is organised as follows: Section 2 discusses the related work on the usage of the performance measures in SDP studies. In Section 3, we provide a complete list of project-specific performance measures, proposed by Sharma et al. in [1]. Section 4 presents the essential requirements for conducting empirical analysis such as utilised datasets, significance test and base-line machine learning (ML) models. In Section 5, we provide detailed empirical results in terms of the proposed measures and discuss the important observations. Section 6 discusses potential threats to the obtained prediction performance. Section 7 concludes the work and provides possible future work.

2. Related Work

In this section, we provide the papers that discuss the key findings from the SDP models in terms of the traditional measures. In this regard, we present the abstract details of two relevant studies that discuss the suitable performance measures for the SDP models. In addition, we also discuss the project-specific performance measures in evaluating the prediction models as suggested in [1].

Since model performance comparison received more attention, in [23], Jiang et al. discussed various traditional performance measures were investigated to find the most suitable candidate for the defect prediction tasks. The study analyses the strengths and weaknesses of the wide variety of numeric evaluation measures such as overall accuracy, error rate, sensitivity, specificity, precision, G-mean, F-measure, J-coefficient, in addition to the graphical summarisation measures such as receiver operating characteristic (ROC) curve, precision and recall (PR) curve, cost curve, and lift chart. The empirical study was conducted using five base-lines on the NASA projects. Since optimising the cost that is being spent on the project and maximising the efficiency of the software verification are the main objectives in developing SDP models, the task for the research practitioners is to minimise the misclassification rate. In this regard, the study [23] does not qualify any best traditional performance measures (this is due to the fact that, in terms of selective performance measures, rarely will one or few models prove to be the best for all possible uses in software quality assessment). However, they concluded that the F-measure offers a balanced consideration of the observed results.

Morasca and Lavazza in [24] conducted a study on choosing the best and relevant portion of the ROC curves, obtained from the predictions of the SDP model. The study proposed a new measure called the ratio of relevant areas (RRA) for evaluating the SDP models by taking only the parts of the ROC curves corresponding to the various values of the threshold. Their work also addresses the shortcomings of the widely used performance measures such as Area Under the Curve (AUC) and the Gini coefficient. However, in summary, their approach provides a theoretical illustration for the use of traditional measures in reducing the misclassification costs of the defect proneness models.

Recently, Sharma et al. in [1] discussed the shortcomings of widely used traditional measures such as F-measure and AUC, and proposed five project-specific performance measures to capture the important observations from the prediction model in terms of the project objectives. The study suggests using an interpretable measure that provides the predictions from the SDP model in terms of cost, service time, and failure, as these are the essential objectives to be accomplished from the prediction model.

Providing an empirical analysis of the prediction models by illustrating their performances in terms of the project-specific attributes is the primary research gap in SDP research. Hence, by extending the study of Sharma et al. in [1], in this work, we provide an empirical evaluation of the widely used defect prediction variants such as WPDP, CPDP, and M-CPDP models in order to validate the use of the project-specific performance measures.

Table 1
The PROMISE projects

Project	Modules	LoC	Defects	%Defects	Project	Modules	LoC	Defects	%Defects
Ant-1.3	125	37,699	20	16.00	Camel-1.4	872	98,080	145	16.63
Ant-1.4	178	54,195	40	22.47	Camel-1.6	965	113,055	188	19.48
Ant-1.5	293	87,047	32	10.92	Jedit-3.2	272	128,883	90	33.09
Ant-1.6	351	113,246	92	26.21	Jedit-4.0	306	144,803	75	24.51
Ant-1.7	745	208,653	166	22.28	Jedit-4.1	312	153,087	79	25.32
Camel-1.0	339	33,721	13	03.83	Jedit-4.2	367	170,683	48	13.08
Camel-1.2	608	66,302	216	35.53	Jedit-4.3	492	202,363	11	02.24

Table 2
The confusion matrix

		Actual values	
		Defective	Clean
Predicted values	Defective	TP	FP
	Clean	FN	TN

3. Project-Specific Measures

In this section, we present the details of the project-specific performance measures, such as percent of perfect cleans, percent of non-perfect cleans, false-omission rate, percent of saved budget, and percent of remaining edits.

The measures PPC and PSB interpret the predictions of the SDP model in terms of cost units, while the PNPC and PRE interpret the predictions in terms of service time units. The measure FOR is used to measure the failure chances from the misclassification of the defective modules. All the measures use the information from the confusion matrix (given in Table 2) among which the measures PPC, PNPC, PSB, and PRE measures utilise an extra attribute called lines of code (LoC) to compute the prediction performances. A detailed explanation of these measures is presented below.

1. Percent of Perfect Cleans (PPC): Since the true negatives (TN) represent the reduced work load, the measure PPC helps in deriving the percentage of reduced work load on the tester. The PPC is derived as the ratio of total TNs to total test instances.

$$\text{PPC} = \frac{|TN|}{|n_t|} \tag{1}$$

Where $|n_t|$ is the number of test instances.

2. Percent of Saved Budget (PSB): Using the measure PPC and an additional attribute called LoC, we estimate the total amount of saved budget in the developing project. The PSB is calculated as:

$$\text{PSB} = \frac{\sum_{i \in TN} SB(LoC_i)}{\sum_{i \in n_t} SB(LoC_i)} \tag{2}$$

Here, we assign a unit cost for servicing each line of code.

3. Percent of Non-Perfect Cleans (PNPC): In contrast to the measure PPC, PNPC is used to represent the percent of work load on the tester from the prediction model. This is because, except for the modules in TN, the tester has to conduct a code walk for all the remaining modules. The measure PNPC is expressed as:

$$PNPC = \frac{|n_t| - |TN|}{|n_t|} \tag{3}$$

4. Percent of Remaining Edits (PRE): Using the measure PNPC and an additional attribute called LoC, we estimate the total service time which is remaining for the tester after utilising the prediction model. Here, for each line of code, we assign a unit time to calculate the service time. This measure is defined below:

$$\text{PRE} = \frac{\sum_{i \in n_t - TN} RE(LoC_i)}{\sum_{i \in n_t} RE(LoC_i)} \tag{4}$$

5. False-Omission Rate (FOR): Unlike other measures, using the measure FOR, we calculate the total failures in the project with the use of SDP models. The major cause of the failures is when the defective module is predicted as clean. Since the total clean modules represents the combination of the false negatives and true negatives, the software may experience failure when the end user triggers a false negative module. Assuming each false negative instance can cause a single failure in the system, the percent of failure instances is measured as:

$$\text{FOR} = \frac{|FN|}{|TN| + |FN|} \tag{5}$$

Note that, the definitions of all these measure are directly taken from the work [1].

4. Study Design

In this section, we provide the details of the utilised datasets (in Section 4.1) and the base-line classifiers (in

Section 4.2). The details of the non-parametric test called Cliff's delta effect size test is provided in Section 4.3. An abstract procedure for the empirical approach is given in Section 4.4.

4.1. Utilised Defects Data

For the empirical analysis, we use publicly available 14 datasets from the PROMISE repository [25]. Each project consist of 24 metrics to describe the software module. Here, the software module can be either a class, method, or a program. We use each software metric as a feature to build the prediction model. A description of the utilised datasets is presented in Table 1.

4.2. Baseline Machine Learners

We perform empirical analysis for three SDP variants using three widely used base-line ML models: SVM, k-NN, and DT. A short description of the utilised baseline ML classifiers is given below.

SVM Classifier: We used a linear kernel function in the training process to compute some extreme data transformations for obtaining the separable data.

k-NN Classifier: The value of k is selected for the k-nearest neighbour (k-NN) model based on 10-fold cross validation. Appropriately, we have chosen k to be 11 after testing the model with various values of k.

Decision Tree Classifier: We have used a general classification and regression trees approach to build the DT classifier.

4.3. Statistical Significance Test

To observe the deviation between the the SDP models, we conduct a non-parametric test called Cliff's delta. This measure provides four levels of effectiveness of the one model over the other models. These levels are given in table 3. The larger value of Cliff's delta indicates the greater effect between the models.

Table 3
Cliff's delta effect size levels [26]

S.No	$\lvert\delta\rvert$	Effectiveness Category
1	$0.000 \leq \lvert\delta\rvert < 0.147$	Negligible
2	$0.147 \leq \lvert\delta\rvert < 0.330$	Small
3	$0.330 \leq \lvert\delta\rvert < 0.474$	Medium
4	$0.474 \leq \lvert\delta\rvert \leq 1.000$	Strong

4.4. Empirical Approach

An empirical evaluation is carried out on each variant of the SDP model. This is because each variant of SDP has different implementation criteria. However, we provide an empirical comparison of the developed models since each variant provides predictions on the same target project dataset. A general training procedure for the three tasks such as WPDP, CPDP, and M-CPDP is given below.

4.4.1. Training and Testing

The WPDP Model: Assume each software project has the availability of local data. Now, we train each base-line ML model on the released versions of the software project [10]. The modules in the latest version (target version or the target project) of the same project are then given as input to the trained WPDP model, in order to observe the predictions.

The CPDP Model: Assume each software project does not have the availability of local data. Now, we train each base-line ML model on the released software projects' defects data [1]. The modules in the newly developed software project (or the target project) are then given as input to the trained CPDP model, in order to observe the predictions.

The M-CPDP Model: Assume each software project has the availability of local data. Also, we assume that defect data for the source projects is available. Now, we train each base-line ML model on the defects data created by augmenting the data from the software project's released versions and the data from the source projects [21]. The modules in the latest version (target version or target project) of the target project are then given as input to the trained M-CPDP model, in order to observe the predictions.

4.4.2. Comparative Approach

To understand the role of project-specific measures in interpreting the performance of the best defect prediction model, we followed the below approach:

Empirical Procedure:

1. First, we use base-line classifiers such as SVM, k-NN, and DT to train the variants of the SDP models such as WPDP, CPDP, and M-CPDP on the PROMISE projects. Each model is evaluated on 10-fold cross validation to observe the mean predictions.
2. Second, we observe the average performances of the trained WPDP, CPDP, and M-CPDP using the project-specific performance measures on each target project.
3. Third, in terms of each base-line classifier, using Cliff's delta effect-size test, we compared the performances of the WPDP, CPDP, and M-CPDP using the project-specific performance measures.

Table 4
Performances of the variants of the SDP models that uses SVM as base classifier

Target Project	PPC			PSC			PNPC			PRE		
	WPDP	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**
Ant-1.3	0.6912	0.8309	0.8223	0.5632	0.6645	0.6423	0.3088	0.1691	0.1777	0.4368	0.3355	0.3577
Ant-1.4	0.6031	0.7428	0.8310	0.6174	0.7187	0.7322	0.3969	0.2572	0.1690	0.3826	0.2813	0.2678
Ant-1.5	0.6931	0.8328	0.8881	0.5652	0.6665	0.6724	0.3069	0.1672	0.1119	0.4348	0.3335	0.3276
Ant-1.6	0.5481	0.6878	0.6592	0.3719	0.4732	0.4613	0.4519	0.3122	0.3408	0.6281	0.5268	0.5387
Ant-1.7	0.5799	0.7196	0.7081	0.3567	0.4580	0.4325	0.4201	0.2804	0.2919	0.6433	0.5420	0.5676
Camel-1.0	0.8082	0.9479	0.9581	0.8163	0.9176	0.9213	0.1918	0.0521	0.0419	0.1837	0.0824	0.0787
Camel-1.2	0.5021	0.6418	0.6614	0.4610	0.5623	0.5747	0.4979	0.3582	0.3386	0.5390	0.4377	0.4253
Camel-1.4	0.7009	0.8406	0.8526	0.6170	0.7183	0.7313	0.2991	0.1594	0.1474	0.3830	0.2817	0.2687
Camel-1.6	0.6589	0.7986	0.8114	0.5921	0.6934	0.7269	0.3411	0.2014	0.1886	0.4079	0.3066	0.2731
Jedit-3.2	0.3819	0.5216	0.4614	0.1756	0.2769	0.2614	0.6181	0.4784	0.5386	0.8244	0.7231	0.7386
Jedit-4.0	0.5339	0.6736	0.5582	0.4492	0.5505	0.5132	0.4661	0.3264	0.4418	0.5508	0.4495	0.4868
Jedit-4.1	0.4826	0.6223	0.4593	0.1877	0.2890	0.2233	0.5174	0.3777	0.5407	0.8123	0.7110	0.7767
Jedit-4.2	0.5620	0.7017	0.6064	0.4619	0.5632	0.4633	0.4380	0.2983	0.3936	0.5381	0.4368	0.5367
Jedit-4.3	0.6660	0.8057	0.9047	0.6877	0.7890	0.8027	0.3340	0.1943	0.0953	0.3123	0.2110	0.1973
Average	0.6008	**0.7405**	0.7273	0.4945	**0.5958**	0.5828	0.3992	**0.2595**	0.2727	0.5055	**0.4042**	0.4172
Cliff's Delta	**0.4692**	0	-	**0.2959**	0	-	**0.4692**	0	-	**0.2959**	0	-

Table 5
Performances of the variants of the SDP models that uses k-NN as base classifier

Target Project	PPC			PSC			PNPC			PRE		
	WPDP	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**	**WPDP**	**CPDP**	**M-CPDP**
Ant-1.3	0.5826	0.7223	0.7615	0.5632	0.6645	0.6766	0.4174	0.2777	0.2385	0.4368	0.3355	0.3234
Ant-1.4	0.5559	0.6956	0.6526	0.5611	0.6624	0.6525	0.4441	0.3044	0.3474	0.4389	0.3376	0.3475
Ant-1.5	0.6817	0.8214	0.8724	0.6911	0.7924	0.8056	0.3183	0.1786	0.1276	0.3089	0.2076	0.1944
Ant-1.6	0.5195	0.6592	0.6767	0.4633	0.5646	0.5852	0.4805	0.3408	0.3233	0.5367	0.4354	0.4148
Ant-1.7	0.5567	0.6964	0.7209	0.5226	0.6239	0.6525	0.4433	0.3036	0.2791	0.4774	0.3761	0.3475
Camel-1.0	0.7703	0.9100	0.9534	0.7205	0.8218	0.8413	0.2297	0.0900	0.0466	0.2795	0.1782	0.1587
Camel-1.2	0.4826	0.6223	0.6102	0.4763	0.5776	0.5614	0.5174	0.3777	0.3898	0.5237	0.4224	0.4386
Camel-1.4	0.7028	0.8425	0.8728	0.6300	0.7313	0.7433	0.2972	0.1575	0.1272	0.3700	0.2687	0.2567
Camel-1.6	0.6127	0.7524	0.7728	0.6021	0.7034	0.7313	0.3873	0.2476	0.2272	0.3979	0.2966	0.2687
Jedit-3.2	0.4318	0.5715	0.5544	0.4230	0.5243	0.5162	0.5682	0.4285	0.4456	0.5770	0.4757	0.4838
Jedit-4.0	0.4686	0.6083	0.6223	0.4714	0.5727	0.5785	0.5314	0.3917	0.3777	0.5286	0.4273	0.4215
Jedit-4.1	0.4661	0.6058	0.5248	0.4719	0.5732	0.5304	0.5339	0.3942	0.4752	0.5281	0.4268	0.4696
Jedit-4.2	0.5220	0.6617	0.6728	0.5219	0.6232	0.6269	0.4780	0.3383	0.3272	0.4781	0.3768	0.3731
Jedit-4.3	0.6137	0.7534	0.8052	0.6052	0.7065	0.7491	0.3863	0.2466	0.1948	0.3948	0.2935	0.2509
Average	0.5691	0.7088	**0.7195**	0.5517	0.6530	**0.6608**	0.4309	0.2912	**0.2805**	0.4483	0.3470	**0.3392**
Cliff's Delta	**0.6429**	0.0561	-	**0.5816**	0.0664	-	**0.6429**	0.0561	-	**0.581**6	0.0664	-

5. Study Results

In this section, we report the observed results of the WPDP, CPDP, and M-CPDP models in terms of the project-specific performance measures.

Tables 4, 5, and 6 provide the performances of the defect prediction models such as WPDP, CPDP, and M-CPDP on the 14 target projects in terms of the measures such as PPC, PSB, PNPC, and PRE, respectively. These tables also provide the results of the Cliff's delta effect-size test. The models such as WPDP, CPDP, and M-CPDP in Table 4 utilised the SVM as a base classifier to observe the predictions on the target datasets, whereas the models in Table 5 utilised the k-NN as a base classifier to observe the predictions on the target datasets. And, the models in Table 6 utilised the DT as a base classifier to observe the predictions on the target datasets.

From Table 4, it is observed that, in the majority of cases, the SVM-based CPDP outperformed the other models in terms of all the performance measures. In particular, the average PPC of the CPDP model has achieved a better value when compared with the other models. Therefore, the testers do not need to visit 74.05% modules to find their defect-proneness. As a consequence, on an average, the savings in the total allocated budget is more using the CPDP model when compared with the other models. Assume a total of 100% allocated budget on the project. Using the CPDP model, the project manager can save up to 59.58% of the budget, whereas with the use of the other models such as WPDP and M-CPDP, the project manager can save only 49.45% and 58.28% of the budget, respectively. On the contrary, since the PNPC is the converse of the measure PPC, the resultant measure PRE also shows its benefits using the CPDP model. Using SVM-based CPDP model, on an average, the testers will have to conduct a code walk on the 40.42% of the total written code. If the testers utilise either WPDP or M-CPDP models, respectively, they have to spend 50.55% and 41.72% of the total written code to observe the defective content. However, the Cliff's delta effect-size test indicating that there is no greater effect between the models such as CPDP and M-CPDP.

Table 6
Performances of the variants of the SDP models that uses DT as base classifier

Target Project	PPC			PSC			PNPC			PRE		
	WPDP	CPDP	M-CPDP	WPDP	CPDP	M-CPDP	WPDP	CPDP	M-CPDP	WPDP	CPDP	M-CPDP
Ant-1.3	0.7037	0.8434	0.8545	0.6849	0.7862	0.7966	0.2963	0.1566	0.1455	0.3151	0.2138	0.2034
Ant-1.4	0.6218	0.7615	0.8314	0.6210	0.7223	0.7313	0.3782	0.2385	0.1686	0.3790	0.2777	0.2687
Ant-1.5	0.6998	0.8395	0.8615	0.7219	0.8232	0.7913	0.3002	0.1605	0.1385	0.2781	0.1768	0.2087
Ant-1.6	0.5817	0.7214	0.7323	0.6077	0.7090	0.6594	0.4183	0.2786	0.2677	0.3923	0.2910	0.3406
Ant-1.7	0.5774	0.7171	0.7089	0.6010	0.7023	0.6947	0.4226	0.2829	0.2911	0.3990	0.2977	0.3053
Camel-1.0	0.8129	0.9526	0.9723	0.7521	0.8534	0.8615	0.1871	0.0474	0.0277	0.2479	0.1466	0.1385
Camel-1.2	0.5130	0.6527	0.6617	0.5311	0.6324	0.6080	0.4870	0.3473	0.3383	0.4689	0.3676	0.3920
Camel-1.4	0.7129	0.8526	0.8633	0.7210	0.8223	0.7966	0.2871	0.1474	0.1367	0.2790	0.1777	0.2034
Camel-1.6	0.6684	0.8081	0.8216	0.6613	0.7626	0.7717	0.3316	0.1919	0.1784	0.3387	0.2374	0.2283
Jedit-3.2	0.3936	0.5333	0.5434	0.4113	0.5126	0.5056	0.6064	0.4667	0.4566	0.5887	0.4874	0.4944
Jedit-4.0	0.5659	0.7056	0.6767	0.5657	0.6670	0.6055	0.4341	0.2944	0.3233	0.4343	0.3330	0.3945
Jedit-4.1	0.4917	0.6314	0.6467	0.5014	0.6027	0.6115	0.5083	0.3686	0.3533	0.4986	0.3973	0.3885
Jedit-4.2	0.5826	0.7223	0.7314	0.6008	0.7021	0.6966	0.4174	0.2777	0.2686	0.3992	0.2979	0.3034
Jedit-4.3	0.6826	0.8223	0.8618	0.6910	0.7923	0.7622	0.3174	0.1777	0.1382	0.3090	0.2077	0.2378
Average	0.6149	0.7546	**0.7691**	0.6194	**0.7207**	0.7066	0.3851	0.2454	**0.2309**	0.3806	**0.2793**	0.2934
Cliff's Delta	**0.6735**	0.1429	-	**0.5205**	-0.1021	-	**0.6735**	0.1429	-	**0.5205**	-0.1021	-

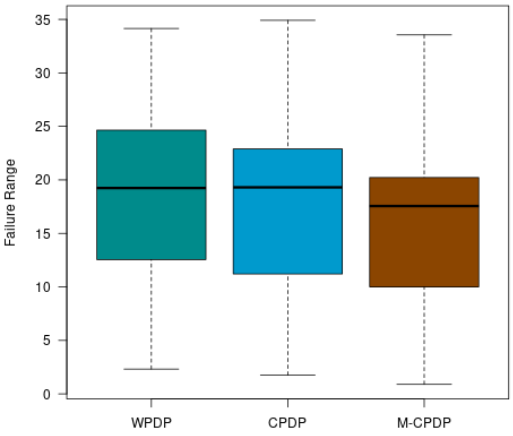

Figure 1: The box-plots representing the observed FOR values on the three models that uses SVM as base classifier

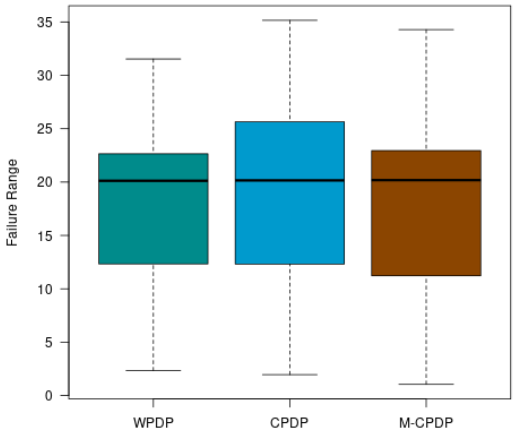

Figure 2: The box-plots representing the observed FOR values on the three models that uses k-NN as base classifier.

From Table 5, it is observed that, in the majority of cases, the k-NN-based M-CPDP outperformed the other models in terms of all the performance measures. In particular, the average PPC of the M-CPDP model has achieved a better value when compared with the other models. Hence, the testers do not need to conduct a code review on 71.95% modules to find their defect-proneness. As a consequence, on an average, the savings in the total allocated budget is more using the M-CPDP model when compared with the other models. For a total of 100% allocated budget on the project, using the M-CPDP model, the project manager can save up to 66.08% of the budget, whereas with the use of the other models such as WPDP and CPDP, the project manager can save 55.17% and 65.30% of the budget, respectively. On the other hand, using k-NN-based M-CPDP model, on an average, the testers will have to conduct a code walk only on the 33.92% of the total written code. If the testers utilise either WPDP or CPDP models, respectively, they have to spend 44.83% and 34.70% of the total written code to observe the defective content. The Cliff's delta effect-size test indicating that there is a negligible but positive effect from the M-CPDP model over the CPDP model.

From Table 6, it is observed that, in the majority of cases, the DT-based M-CPDP outperformed the other models in terms of the performance measures such as PPC and PNPC. While the model CPDP performed better than the other models in terms of measures such as

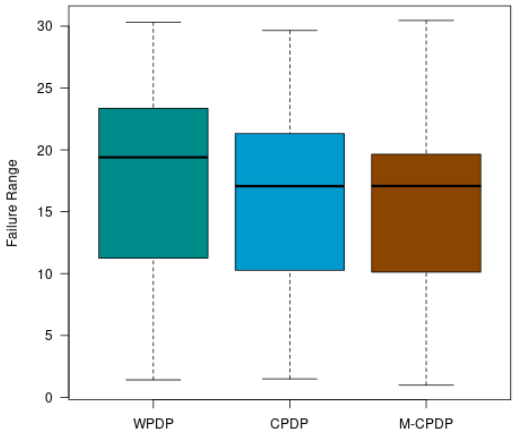

Figure 3: The box-plots representing the observed FOR values on the three models that uses DT as base classifier.

PSB and PRE. In particular, the average PPC of the M-CPDP model has achieved a better value when compared with the other models. Hence, the testers do not need to conduct a code review on 76.91% modules to find their defect-proneness. In contrast, on an average, the savings in the total allocated budget is more using CPDP model, when compared with the other models. On a total of 100% allocated budget on the project, using CPDP model, the project manager can save up to 72.07% of the budget, whereas with the use of the other models such as WPDP and M-CPDP, the project manager can save 61.94% and 70.66% of the budget, respectively. On the other hand, using the DT-based CPDP model, on an average, the testers will have to conduct a code walk only on the 27.93% of the total written code. If the testers utilise either WPDP or M-CPDP models, respectively, they have to spend 38.06% and 29.34% of the total written code to observe the defective content. The Cliff's delta effect-size test indicating that there is a negligible but positive effect from the M-CPDP model over the CPDP model in terms of PPC and PNPC measures. In terms of PSC and PRE, the Cliff's delta effect-size test indicating that there is a negligible but negative effect from the M-CPDP model over the CPDP model.

The box-plots in Figures 1, 2, and 3 represent the chances of failure incidents as a result of SVM, k-NN, and DT-based SDP models in the target projects. From Figures 1, 2, and 3, it is observed that, the median failure incidents are fewer using the M-CPDP model (which is trained using three classifiers) when compared with the other models. Since any software project should least expect a misclassifications from the prediction model, the M-CPDP model may suit well in real-time testing environments.

5.1. Discussion

Any project seeks benefits from the prediction model, hence, achieving its goal is of primary importance to the researcher. The major obstacle in selecting the model is obtaining a trade-off between the obtained performances from the various prediction models. In Section 5, we observe that, on the majority of target projects, the CPDP model has achieved its better performance in terms of PPC and PNPC. This shows that the CPDP model is good at predicting clean modules more accurately. However, surprisingly, the M-CPDP model has shown its strength in terms of budget savings, minimal service time, and more importantly, minimal failure incidents on the majority of the target projects. Hence, even though the CPDP model is better in terms of PPC and PNPC, the M-CPDP is better in terms of all the performance measures.

From Tables 4, 5, and 6, and Figures 1, 2, and 3, it is observed that, among all the baselines, after 10-fold cross validation, the decision tree-based M-CPDP model has achieved maximum budget savings, minimal service time, and minimal failure incidents on the majority of the target projects.

6. Threats to Validity

Variation in the observed performances at various working environments is common in the empirical research. In this section, we present the factors that may affect the observed performances.

Internal Validity:

The observed performances are based on the usage of a few base-line ML models, and the M-CPDP model has shown its strength using majority of the baselines. However, implementing the other baselines such as logistic regression, neural networks, ensemble models, etc. on the other widely used repositories such as NASA, AEEEM, ReLink, etc. is the major threat that may hinder the final performance of the M-CPDP model.

External Validity:

For the purpose of knowing the best model that is suitable for the real-time testing environments, we performed an empirical analysis on only three variants of SDP using the project-specific performance measures. The generalised conclusions can be made when conducting the empirical analysis on the other variants of the SDP such as MPDP, *pair-wise* CPDP, just-in time software defect prediction (JIT-SDP), and heterogeneous defect prediction (HDP).

7. Conclusion and Future Work

The research on proposing the software defect prediction (SDP) models is intended to diminish the workload on the tester by providing intelligent decisions on the defect-proneness of the newly developed software module. Hence, the objective of the SDP models is to decrease the time, cost, and manpower that are being spent on the software project. Inherently, the task of the SDP models is also to reduce the risk of misclassification (in particular, false negatives). In this regard, Sharma et al. in [1] proposed project-specific performance measures such as percent of perfect-cleans, percent of saved budget, percent of non-perfect cleans, percent of remaining edits, and false omission rate to interpret the obtained results in terms of the project objectives. Since it is important for the software engineering researcher to provide a better prediction model, it is necessary to interpret the results in terms of the project-specific objectives.

Extending the work of Sharma et al.[1], in this paper, we conducted an empirical analysis of the interpretation of the project-specific performance measures on the variants of SDP such as WPDP, CPDP, and *mixed*-CPDP. With the empirical analysis of the PROMISE projects, we conclude that the models such as CPDP and M-CPDP have achieved significantly better performances in terms of all the measures than the WPDP model. Among CPDP and M-CPDP, we observe that the number of failure incidents is lower with the use of the M-CPDP models. Also, on the majority of the target projects, the software managers may benefit from the use of M-CPDP models in terms of maximum savings in the allocated budget and minimal time required to service the code. Hence, we recommend using M-CPDP models in real-time testing environments.

Possible future research directions from this work include: (1) conducting a large-scale empirical analysis of all the variants of SDP on widely-used defect repositories such as PROMISE, NASA, AEEEM, ReLink, GitHub, etc. using the project-specific performance measures. (2) estimating the real-time feasibility of the M-CPDP model.

References

[1] U. S. B., R. 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. URL: https://www.sciencedirect.com/science/article/pii/S0164121222001984. doi:https://doi.org/10.1016/j.jss.2022.111522.

[2] V. R. Basili, L. C. Briand, W. L. Melo, A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 22 (1996).

[3] V. U. B. Challagulla, F. B. Bastani, R. Paul, Empirical Assessment of Machine Learning based Software Defect Prediction Techniques, 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (2005) 263–270.

[4] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (2008) 485–496.

[5] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Systems with Applications 36 (2009) 7346–7354.

[6] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical Software Engineering 17 (2012) 531–577.

[7] B. Ghotra, S. McIntosh, A. E. Hassan, A large-scale study of the impact of feature selection techniques on defect classification models, in: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 146–157.

[8] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, A. De Lucia, A Developer Centered Bug Prediction Model, IEEE Transactions on Software Engineering (2018).

[9] L. Kumar, S. K. Sripada, A. Sureka, S. K. Rath, Effective fault prediction model developed using least square support vector machine (lssvm), Journal of Systems and Software 137 (2018) 686–712.

[10] U. S. Bhutamapuram, R. Sadam, With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique, Journal of King Saud University-Computer and Information Sciences (2021). URL: https://doi.org/10.1016/j.jksuci.2021.09.010.

[11] J. Xu, F. Wang, J. Ai, Defect prediction with semantics and context features of codes based on graph representation learning, IEEE Transactions on Reliability (2020).

[12] L. C. Briand, W. L. Melo, J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE transactions on Software Engineering 28 (2002) 706–720.

[13] T. M. Khoshgoftaar, E. B. Allen, J. Deng, Using regression trees to classify fault-prone software modules, IEEE Transactions on Reliability (2002).

[14] T. Menzies, J. DiStefano, A. Orrego, R. Chapman, Assessing predictors of software defects, in: Proc. Workshop Predictive Software Models, 2004.

[15] K. O. Elish, M. O. Elish, Predicting defect-prone software modules using support vector machines, Journal of Systems and Software 81 (2008) 649–660.

[16] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, in: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 91–100.

[17] I. H. Laradji, M. Alshayeb, L. Ghouti, Software defect prediction using ensemble learning on selected features, Information and Software Technology 58 (2015) 388–402.

[18] L. Kumar, S. Misra, S. K. Rath, An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes, Computer standards & interfaces 53 (2017) 1–32.

[19] S. Hosseini, B. Turhan, M. Mäntylä, A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction, Information and Software Technology 95 (2018) 296–312.

[20] H. Chen, X.-Y. Jing, Z. Li, D. Wu, Y. Peng, Z. Huang, An empirical study on heterogeneous defect prediction approaches, IEEE Transactions on Software Engineering (2020).

[21] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Transactions on Software Engineering 44 (2017) 811–833.

[22] C. Ni, X. Chen, F. Wu, Y. Shen, Q. Gu, An empirical study on pareto based multi-objective feature selection for software defect prediction, Journal of Systems and Software 152 (2019) 215–238.

[23] Y. Jiang, B. Cukic, Y. Ma, Techniques for evaluating fault prediction models, Empirical Software Engineering 13 (2008) 561–595.

[24] S. Morasca, L. Lavazza, On the assessment of software defect prediction models via roc curves, Empirical Software Engineering 25 (2020) 3977–4019.

[25] J. Sayyad Shirabad, T. Menzies, The PROMISE Repository of Software Engineering Databases, School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL: http://promise.site.uottawa.ca/SERepository.

[26] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions., Psychological bulletin 114 (1993) 494.