APPENDICES

for

" Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction"

Chao Ni, Xin Xia, David Lo, Xiang Chen, and Qing Gu

APPENDIX A
AN EXAMPLE ON HOW TO CALCULATE EPMS

For a better understanding of how these recently proposed methods calculating EPMs (i.e., IFA, $PII@L$ and $CostEffort@L$), we describe the calculating process with an example shown in Table 1.

In Table 1, suppose we have a testing dataset, which contains eight instances. The first column represents the original order of these instances. These instances are numbered from 1 to 8 shown in the second column. The LOC of instances and the class label of instances are presented in the following two columns. Then, the predicted score and predicted results of each method are given in the next six columns. Finally, $score/LOC$ is listed in the last column.

- **Sorting strategy for state-of-the-art CPDP methods:** The state-of-the-art CPDP methods sort the testing instances in descending order of $score$ (i.e., the probability of defect-prone outputted by prediction model).
- **Sorting strategy for EASC method:** EASC uses different sorting strategies on testing instances when calculating different types of performance measures. In particular, when calculating EMPs, EASC firstly predicts testing instances, and divides all instances into two groups: Group_Defective and Group_Clean. For instances in Group_Defective, the probability of defect-proneness of each instance is larger than 0.5, while instances in Group_clean have less than 0.5 probability of defect-proneness. After that, all the instances in the two groups will be sorted by $score/LOC$, respectively. Finally, the two groups are combined together. In particular, these instances in Group_Clean will be appended at the end of Group_Defective. The results are shown in Figure 1.

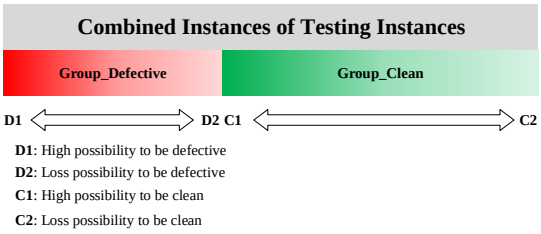


Fig. 1: Sorting Strategy for EASC.

In the above figure, different colors represent different types of instances: defective instance and clean instance. Besides, instances in each group have different color depth. The color depth indicates the defect density of each instance, which to some extent indicates the priority of inspecting these instances.

- **Sorting strategy for ManualDown/ManualUp methods:** Zhou et al. proposed two unsupervised methods in the scenario of cross project defect prediction: ManualDown and ManualUp. For the simplicity of presentation, let m be a module in the testing data, $SizeMetric$ be a module size metric, and $R(m)$ be the predicted risk value of the module m. Formally, the ManualDown method is $R(m) = SizeMetric(m)$, while the ManualUp method is $R(m) = 1/SizeMetric(m)$.
For a given testing data, ManualDown considers a larger module as more defect-prone. However, ManualUp considers a smaller module as more defect-

- *Chao Ni and Qing Gu are with State Key Laboratory for Novel Software Technology, Nanjing University, No.163 Xianlin Rd, Qixia District, Nanjing, Jiangsu, 210023, China,*
  E-mail: jacknichao920209@gmail.com, quq@nju.edu.cn
- *Xin Xia is with the Faculty of Information Technology, Monash University, Melbourne, Australia.*
  E-mail: xin.xia@monash.edu
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore.*
  E-mail: davidlo@smu.edu.sg
- *Xiang Chen is with the School of Information Science and Technology Science, Nantong University, China.*
  E-mail: xchencs@ntu.edu.cn
- *Qing Gu and Xin Xia are the corresponding authors.*

Manuscript received ; revised.

TABLE 1: An example with 8 testing instances for calculating EPMs

OO	No.	LOC	Actual	Score			Predicted			Score / LOC
				CPDP	EASC	ManualUp	CPDP	EASC	ManualUp	
				Score 1	Score 2	Score 3	Predict 1	Predict 2	Predict 3	
1	①	50	0	0.43	0.22	1/50	0	0	1	4.40E-03
2	②	180	1	0.86	0.97	1/180	1	1	1	5.44E-03
3	③	100	0	0.495	0.12	1/100	0	0	1	1.20E-03
4	④	500	0	0.45	0.33	1/500	0	0	0	6.60E-04
5	⑤	758	1	0.34	0.89	1/758	0	1	0	1.17E-03
6	⑥	1000	0	0.33	0.12	1/1000	0	0	0	1.20E-04
7	⑦	80	1	0.78	0.78	1/80	1	1	1	9.75E-03
8	⑧	210	0	0.32	0.48	1/210	0	0	0	2.29E-03

Notes: (1) OO: Original Order; (2) No.: Instances Number.

prone. Besides, in their work, they use LOC as the $SizeMetric$. Therefore, when calculating NPMs: ManualDown sorts all testing instances in descending order by LOC. Then, ManualDown classifies these instances sorted at the top 50% as defective ones, while ManualDown classifies these instances sorted at the bottom 50% as clean ones.

When calculating EPMs: ManualUp sorts all testing instances in descending order by $1/LOC$. Then, ManualUp classifies these instances sorted at the top 50% as defective ones, while ManualUp classifies these instances sorted at the bottom 50% as clean ones.

In this section, we pay more attention on ManualUp since Zhou et al. suggested ManualUp should be treated as base method when considering EPMs.

According to the sorting strategies of different methods, we give the re-sorted testing data for three methods independently.

- Order of CPDP methods:

Actual Results	①②③④⑤⑥⑦⑧
Sorted Instances order	②⑦③④①⑤⑥⑧
Prediction Results	①②③④⑤⑥⑦⑧
20% of Inspection Effort	②⑦③

- Order of EASC method:

Actual Results	①②③④⑤⑥⑦⑧
Sorted Instances order	⑦②⑤①⑧③④⑥
SCORE / LOC	9.75e-03,5.44e-03,1.17e-03, 4.40e-03,2.29e-03,1.20e-03, 6.60e-04, 1.20e-04
Prediction Results	①②③④⑤⑥⑦⑧
20% of Inspection Effort	⑦②

- Order of ManualUp method:

Actual Results	①②③④⑤⑥⑦⑧
Sorted Instances order	①⑦③②⑧④⑤⑥
1 / LOC	1/50,1/80,1/100,1/180, 1/210,1/500,1/758,1/1000
Prediction Results	①②③④⑤⑥⑦⑧
20% of Inspection Effort	①⑦③②

Therefore, based on the above results, the results of IFA and $PII@20\%$ for these methods are listed as follows:

EPMs	CPDP	EASC	ManualUp
IFA	1	0	1
PII@20%	3/8	2/8	4/8
CostEffort@20%	2/3	2/3	2/3

From the above tables, in terms of IFA, $PII@20\%$ and $CostEffort@20\%$, we find that EASC can achieve a good trade-off performance. In particular, when compared with CPDP and ManualUp, EASC can not only have less false alarm since its IFA is small, but it also has less context switch since its $PII@20\%$ is still small. Besides, all these methods achieve same performance of $CostEffort@20\%$.

APPENDIX B
EXPERIMENTAL SUBJECTS

In our experimental studies, we evaluate CPDP methods on four publicly available datasets, which are also used in [1]–[30]. Table 2 gives an overview of these datasets. The first column reports the name of the group. The second to sixth columns, respectively, report the target project, the project type, the programming language, the brief description, the number of instances contained (one instance corresponding to one module), and the percentage of defective instances. The last column reports the effort metric name in different datasets, which represents the proxy of effort when inspecting whether an instance has a defect or not. Note that, in our experiment, we only consider the **strict cross-project defect prediction scenario**. That means all other versions of a specific product will be excluded for training. Therefore, "Eclipse", investigated in [31], is not considered since the dataset only has one project with different versions.

NASA. The first dataset is the preprocessed version of the NASA Metrics Data Program data provided by Shepperd et al. [32]. The dataset contains information about 12 products from 6 projects. We use the preprocessed version since Shepperd et al. resolved the problems with the consistency of the originally published MDP data noted by Gray et al. [33]. There are 17 static source code metrics used in the dataset. However, information about how the defect labels were created is not available. In our experiment, we use all 12 products from this data set and refer to this dataset as NASA.

AEEEM. The second dataset was shared by D'Ambros et al. [34] and contained data about five Java products from different projects. Sixty one software metrics are considered, including static product metrics, process metrics like the number of defects in previous releases, the entropy of code changes, and source code churn, as well as the weighted churn and of source code metrics (WCHU) and linearly decayed entropy of source code metrics (LDHH). The defect labels were extracted from the Issue Tracking System (ITS)

of the projects. In our experiment, we use all five original products from this dataset and refer to this dataset as AEEEM.

RELINK. The third dataset was shared by Wu et al. [35] and contains defect information about three products from different projects. The dataset has 60 static product metrics and three different defect labels for each module: 1) golden, with manually verified and not automatically labelled defect labels; 2) relink, with defect labels generated with their proposed approach; and 3) traditional heuristic, with an SCM comment based labelling. In our experiment, we use all 3 products with the golden set labelling from this dataset and refer to this dataset as RELINK.

PROMISE. The fourth data set was shared by Jureczko and Madeyski [36]. In our experiment, we use 65 product versions of 32 projects which are provided by Herbold et at. [1]. As for metrics, they collected 20 static product metrics for Java classes, as well as the number of defects that were found in each class. According to [36], all the labels are extracted from the source code management system using a regular expression. Notice that in Herbold et al.'s work, the dataset is named as JURECZKO. In our experiment, we use all 32 products from this dataset and refer to this dataset as PROMISE since it is referred as such in previous works [2], [31], [37], [38].

TABLE 2: The Characteristics of Studied Datasets.

Dataset	Project	Lang.	Description	#Instances	%Defective	Effort Metric
NASA	CM1	C	A spacecraft instrument	344	12.21%	LOC_EXECUTABLE (The number of lines of executable code for a module)
	JM1		A real time C project	9593	18.34%	
	KC1	C++	A storage management system for ground data	2096	15.51%	
	KC3	Java		200	18.00%	
	MC1		A combustion experiment	9277	0.73%	
	MC2	C	A video guidance system	127	34.65%	
	MW1		A zero gravity experiment related to combustion	264	10.23%	
	PC1		A flight software from an earth orbiting satellite	759	8.04%	
	PC2		A dynamic simulator for attitude control systems	1585	1.01%	
	PC3		A flight software from an earth orbiting satellite	1125	12.44%	
	PC4			1399	12.72%	
	PC5			17001	2.96%	
AEEEM	equinox	Java	An OSGi R4 core framework implementation	324	39.81%	numberOfLinesOfCode (Number of lines of code)
	elipse jdt		The Java infrastructure of the Java IDE	997	20.66%	
	lucene		A text search engine library	691	9.26%	
	mylyn		A task management plugin	1862	13.16%	
	pde		A plug-in development environment	1497	13.96%	
RELINK	Apapce Httpd	Java	An open-source HTTP server	194	50.52%	CountLineCode (Number of lines of code)
	OpenIntents Safe		An open intents library	56	39.29%	
	Zxing		A barcode image-processing library	399	29.57%	

continued on the next page

TABLE 2: The Characteristics of Studied Datasets. – continued from previous page

Dataset	Project	Lang.	Description	#Instances	%Defective	Effort Metric
PROMISE	ant 1.3, 1.4 1.5,1.6, 1.7	Java	A build management system	125-745	10.92%-26.21%	loc(Number of lines of code)
	arc		Academic software project developed by 8th or 9th semester computer science students	234	11.54%	
	berek			43	37.21%	
	camel 1.0, 1.2, 1.4, 1.6		A versatile integration framework	339-965	3.83%-35.53%	
	ckjm		A tool for collecting Chidamber and Kemerer metrics	10	50.00%	
	e-learning		Academic software project developed by 8th or 9th semester computer science students	64	7.81%	
	forrest 0.7			29	17.24%	
	ivy 1.1, 1.4, 2.0		A dependency manager	111-352	6.64%-56.76%	
	jedit 3.2, 4.0, 4.1, 4.2, 4.3		A text editor	272-492	2.24%-33.09%	
	kalkulator		Academic software project developed by 8th or 9th semester computer science students	27	22.22%	
	log4j 1.0, 1.1, 1.2		A logging utility	109-205	25.19%-92.20%	
	lucene 2.0, 2.2, 2.4		A text search engine library	195-340	46.67%-59.71%	
	nieruchomosci		Academic software project developed by 8th or 9th semester computer science students	27	37.04%	
	pbeans 1.0, 2.0		An object/relational database mapping framework	26-51	19.61%-76.92%	
	pdftranslator		Academic software project developed by 8th or 9th semester computer science students	33	45.45%	
	poi 1.5, 2.0, 2.5, 3.0		API for Office Open XML standards	237-442	11.78%-64.42%	
	redaktor		A decentralized content management system	176	15.34%	
	serapion		Academic software project developed by 8th or 9th semester computer science students	45	20.00%	
	skarbonka			45	20.00%	
	sklebagd			20	60.00%	
	synapse 1.0, 1.1, 1.2		A high-performance Enterprise Service Bus	157-256	10.19-33.59%	
	systemdata		Academic software project developed by 8th or 9th semester computer science students	65	13.85%	
	szybkafucha			25	56.00%	
	termoproject			42	30.95%	
	tomcat		A Web server	858	8.97%	
	velocity 1.4, 1.5, 1.6		A template language engine	196-229	34.06%-75.00%	
	workflow		Academic software project developed by 8th or 9th semester computer science students	39	51.28%	
	wspomaganiepi			18	66.67%	
	xalan 2.4, 2.5, 2.6, 2.7		An XSLT processor	723-909	15.21%-98.79%	
	xerces 1.0, 1.2, 1.3, 1.4		An XML processor	162-588	15.23%-74.32%	
	zuzel		Academic software project developed by 8th or 9th semester computer science students	29	44.83%	

APPENDIX C
BRIEF INTRODUCTION TO FOUR STATE-OF-THE-ART SUPERVISED METHODS

Cruz and Ochimizu [39] proposed to apply a power transformation to the metric data and then standardize it. The power transformation is based on the logarithm and the observation that software metrics, especially the size and complexity, often follow exponential distributions, which is the same as what Turhan et al. [40] do for the treatment of the data. Besides, they only consider training product as a reference. They conducted an experiment on seven Java projects and found that the CPDP methods with standardization can achieve better performance than the corresponding CPDP methods without standardization.

Turhan et al. [40] proposed Burak filter to first transform the metric data with the logarithm and then applied a relevancy filter to the available training data based on the k (i.e., 10) nearest instances algorithm. Through the relevancy filter, the k nearest instances for each instance in the target data are selected. Notice that repeat selected instance will be used only once. They conducted an experiment on seven products from the NASA dataset and all three products from the SOFTLAB dataset and found that the Burak filter can achieve better performance on $recall$ and pf for both WPDP scenario and CPDP scenario.

Menzies et al. [41] created a local model through clustering of the training data with the WHERE algorithm and afterwards classification of the results with the WHICH rule learning algorithm. Separate WHICH rules are created for each cluster to create local models. In addition to WHICH, random forest is used in this paper due to its better performance. They conducted an experiment on seven products from the PROMISE dataset and observed a further gain in terms of the median over the method using all data.

Watanabe et al. [42] proposed to compensate differences between products through a standardization technique that rescales the data. In a scenario with only one candidate product as training data, they proposed to use this product as the reference for the standardization of the target data. This shall increase the homogeneity between the target product and the candidate product. As a formula for standardization, they proposed to multiply each metric value of the target product with the mean value of the candidate product and divided this by the mean of the target product itself. They conducted an experiment on two projects mined by themselves using seven metrics and bug labels based on the comments in the version control system logs. They observed a little improvement in $recall$ (e.g., 15%) and $precision$ (e.g., 2%).

APPENDIX D
MORE STATISTICAL COMPARISON RESULT BETWEEN SUPERVISED AND UNSUPERVISED METHODS.

As suggested by Zhou et al., we should treat ManualDown (ManualUp) as baseline method when inspection effort is unlimited (limited). Therefore, for the convenience of reading, we present a part of statistical results between supervised methods and unsupervised methods in main article. For example, when considering NPMs, we only present the statistical test between ManualDown and supervised methods, and when considering EPMs, we only present the statistical test between ManualUp and supervised methods. In this section, for comprehensively understanding the difference between supervised and unsupervised methods, we list more statistical information.

TABLE 3: Comparisons between EASC and **ManualUp** on four datasets in terms of **Non-effort-aware Performance Measures**.

Measures	Datasets	EASC	ManualUp	ManualDown
$F1-score^{\uparrow}$	AEEEM	**0.32±0.02(L)***	013±0.00	0.39±0.03
	NASA	**0.26±0.01(L)****	009±0.01	0.27±0.02
	PROMISE	**0.28±0.02(S)***	022±0.03	0.50±0.03
	RELINK	0.67±0.02	024±0.00	0.64±0.01
$AUC^{\uparrow}$	AEEEM	**0.75±0.00(L)****	027±0.00	0.73±0.00
	NASA	**0.77±0.01(L)*****	026±0.01	0.74±0.01
	PROMISE	**0.73±0.01(L)*****	027±0.01	0.73±0.01
	RELINK	0.79±0.01	026±0.01	0.74±0.01
$False\ Alarm^{\downarrow}$	AEEEM	**0.07±0.01(L)****	056±0.00	0.43±0.00
	NASA	**0.07±0.00(L)*****	053±0.00	0.46±0.00
	PROMISE	**0.07±0.00(L)*****	061±0.01	0.38±0.01
	RELINK	0.23±0.05	064±0.00	0.33±0.01

Notes: (1) *** means $p < 0.001$, ** means $p < 0.01$, * means $p < 0.05$.
(2) L/M/S: Large/Medium/Small effect size according to Cliff's delta.
(3) '↓' indicates 'the smaller the better'; '↑' indicates 'the larger the better'.

Table 3 and Table 4 present the average results and statistical analysis results of supervised and unsupervised methods when NPMs and EPMs are considered, respectively. In both of the two tables, the first column lists the performance measures. The second column lists the datasets we experiment on. In the following column, the average performance values (i.e., the mean performance values) of EASC are given. Then we list the average performance of unsupervised methods. Notice that the results listed in the columns filled with grey is used for comparison. For example, Table 3 mainly lists the comparison of EASC and ManualUp in terms of NPMs. But for convenience, we also list the average result of ManualDown in the last column. Besides, for columns of supervised methods, we use different ways to present the statistical analysis results. In particular, the cells are in **bold** if the supervised method is significantly superior to the unsupervised method, the cells are in underline if the supervised method is significantly inferior to the unsupervised method. Furthermore, we use different number of symbol "*" to represent the level of p-value (i.e., *** means $p < 0.001$, ** means $p < 0.01$, * means $p < 0.05$). The effect sizes are also indicated using the "L/M/S" character, which correspondingly represents the Large/Medium/Small effect size according to Cliff's delta.

From the results shown in Table 3 and Table 4, we make the following observations:

(1) **NPMs**. EASC can statistically significantly outperform ManualUp with a large improvement for almost all cases in terms of $F1\text{-}score$, AUC and $False\ Alarm$.
(2) **EPMs**. ManualDown can statistically significantly outperform EASC in most cases in terms of $PII@L$, while EASC can statistically significantly outperform ManualDown for almost all cases in terms of $CostEffort@L\%$ and P_{opt}. In terms of IFA, ManualDown and EASC obtain similar performance without statistical significance except for PROMISE.

Besides, we also conduct experiments on the comparisons of the four state-of-the-art supervised methods and two unsupervised methods. The results are listed in the Table 5 and Table 6.

From Table 5 and Table 6, we find that:

(1) **NPMs**. The four state-of-the-art supervised methods can statistically significantly outperform ManualUp with a large improvement for almost all cases in terms of $F1\text{-}score$, AUC and $False\ Alarm$.
(2) **EPMs**. ManualDown can statistically significantly outperform the four state-of-the-art supervised methods in most cases in terms of IFA and $PII@L$. In terms of $CostEffort@L\%$ and P_{opt}, four state-of-the-art supervised methods statistically significantly outperform ManualDown in most cases.

APPENDIX E
THE DISTRIBUTION OF INSTANCE INSPECTION EFFORT AND INSTANCE QUALITY ON FOUR DATASETS

In Table 7 to Table 10, the first column lists the dataset name. The second column lists the name of project. In the following five columns, we list the percentage of defective instances in the top sorted instances based on inspection effort. In Zhou et al.'s method, they used 50% as the classification threshold. We list the results of five different thresholds (i.e., 10%, 20%, 30%, 40% and 50%). Next, we list the total number of defective instances in each project. In the following five columns, we list the percentage of effort in the top sorted instances based on inspection effort. Notice that the proxy of inspection effort can be found in **APPENDIX** B. Followed that, we list the total number of inspection and instances in each project respectively. The last column shows the distribution of defective modules by using ManualDown method via visualization technology. Notice that all the instances in each dataset are represented as stripes, and sorted by their inspection effort in descending order from the left side to the right side. In each distribution figure, the pink stripes represent the defective instances, and the green stripes represent the non-defective instances.

For each project, we find that the majority of defective instances (i.e., more than 70%) will be ranked on the top when sorting the instances according to its inspection effort in descending order and inspect the top 50%. In particular, we find that ManualDown can find at least 67%, 68%, 64% and 44% defective instances of total defective instances on AEEEM, RELINK, NASA and PROMISE respectively. However, we need to spend 87%, 91%, 77% and 80% of total inspection effort on AEEEM, RELINK, NASA and PROMISE

TABLE 4: Comparisons between EASC and **ManualDown** on four datasets in terms of **Effort-aware Performance Measures**.

Measures	Dataset	EASC	ManualDown	ManualUp	Measures	Dataset	EASC	ManualDown	ManualUp
$IFA^{\downarrow}$	AEEEM	1±2	1±2	30±417	$P_{opt}@20\%^{\uparrow}$	AEEEM	**0.73±0.02(L)****	0.22±0.03	0.65±0.00
	NASA	5±50	1±16	1268±7251939		NASA	**0.62±0.02(L)***	0.41±0.04	0.49±0.04
	PROMISE	6±118(M)***	1±2	20±538		PROMISE	**0.66±0.08(L)*****	0.20±0.08	0.63±0.04
	RELINK	1±2	0±0	8±1		RELINK	0.74±0.02	0.32±0.08	0.63±0.00
$PII@20\%^{\downarrow}$	AEEEM	0.08±0.00(L)**	0.02±0.00	0.69±0.00	$CostEffort@20\%^{\uparrow}$	AEEEM	**0.18±0.01(L)****	0.05±0.00	0.26±0.00
	NASA	0.07±0.00(L)*	0.03±0.00	0.54±0.02		NASA	**0.20±0.01(L)***	0.10±0.00	0.18±0.02
	PROMISE	0.11±0.00(L)***	0.03±0.00	0.68±0.01		PROMISE	**0.17±0.01(L)*****	0.08±0.00	0.27±0.02
	RELINK	0.22±0.01	0.04±0.00	0.68±0.00		RELINK	0.33±0.00	0.09±0.00	0.28±0.00
$PII@1000^{\downarrow}$	AEEEM	0.02±0.00(L)**	0.00±0.00	0.19±0.02	$CostEffort@1000^{\uparrow}$	AEEEM	**0.04±0.00(L)***	0.00±0.00	0.08±0.00
	NASA	0.04±0.00	0.02±0.00	0.25±0.02		NASA	0.11±0.02	0.05±0.00	0.08±0.02
	PROMISE	0.06±0.01(L)***	0.03±0.00	0.35±0.04		PROMISE	0.05±0.00	0.05±0.01	0.15±0.01
	RELINK	0.12±0.00	0.02±0.00	0.48±0.05		RELINK	0.22±0.04	0.06±0.00	0.19±0.01
$PII@2000^{\downarrow}$	AEEEM	0.02±0.00(L)**	0.00±0.00	0.26±0.02	$CostEffort@2000^{\uparrow}$	AEEEM	**0.05±0.00(L)***	0.01±0.00	0.12±0.01
	NASA	0.07±0.01	0.05±0.01	0.39±0.05		NASA	0.16±0.02	0.11±0.02	0.11±0.02
	PROMISE	0.10±0.04(L)***	0.06±0.03	0.45±0.05		PROMISE	0.07±0.01(S)*	0.08±0.02	0.18±0.02
	RELINK	0.18±0.00	0.05±0.00	0.61±0.07		RELINK	0.32±0.06	0.12±0.03	0.24±0.00

Notes: (1) *** means $p < 0.001$, ** means $p < 0.01$, * means $p < 0.05$.
(2) L/M/S: Large/Medium/Small effect size according to Cliff's delta.
(3) '↓' indicates 'the smaller the better'; '↑' indicates 'the larger the better'.

TABLE 5: Comparisons among supervised methods and **ManualUp** on four datasets in terms of non-effort-aware performance measures.

Measures	Dataset	CamargoCruz09-DT	Menzies11-RF	Turhan09-DT	Watanabe08-DT	ManualUp	ManualDown
$F1-score^{\uparrow}$	AEEEM	**0.31±0.01(L)***	**0.27±0.02(L)***	**0.27±0.00(L)***	**0.30±0.00(L)****	0.13±0.00	0.39±0.03
	NASA	0.09±0.01	0.12±0.01	0.16±0.01	0.11±0.00	0.09±0.01	0.27±0.02
	PROMISE	**0.37±0.02(L)*****	**0.33±0.03(M)*****	**0.36±0.03(M)*****	**0.37±0.01(L)*****	0.22±0.03	0.50±0.03
	RELINK	0.54±0.00	0.59±0.03	0.53±0.03	0.49±0.03	0.24±0.00	0.64±0.01
$AUC^{\uparrow}$	AEEEM	**0.60±0.01(L)****	**0.58±0.00(L)****	**0.53±0.00(L)****	**0.59±0.00(L)****	0.27±0.00	0.73±0.00
	NASA	**0.70±0.01(L)*****	**0.53±0.00(L)*****	**0.62±0.00(L)*****	**0.67±0.01(L)*****	0.26±0.01	0.74±0.01
	PROMISE	**0.58±0.01(L)*****	**0.59±0.01(L)*****	**0.59±0.01(L)*****	**0.59±0.01(L)*****	0.27±0.01	0.73±0.01
	RELINK	0.65±0.00	0.68±0.01	0.63±0.01	0.60±0.02	0.26±0.01	0.74±0.01
$False\ Alarm^{\uparrow}$	AEEEM	**0.06±0.00(L)****	**0.04±0.00(L)****	**0.13±0.01(L)****	**0.11±0.00(L)****	0.56±0.00	0.43±0.00
	NASA	**0.01±0.00(L)*****	**0.03±0.00(L)*****	**0.05±0.00(L)*****	**0.02±0.00(L)*****	0.53±0.00	0.46±0.00
	PROMISE	**0.20±0.01(L)*****	**0.13±0.01(L)*****	**0.18±0.01(L)*****	**0.26±0.02(L)*****	0.61±0.01	0.38±0.01
	RELINK	0.21±0.01	0.20±0.00	0.17±0.02	0.17±0.01	0.64±0.00	0.33±0.01

Notes: (1) *** means $p < 0.001$, ** means $p < 0.01$, * means $p < 0.05$.
(2) L/M/S: Large/Medium/Small effect size according to Cliff's delta.
(3) '↓' indicates 'the smaller the better'; '↑' indicates 'the larger the better'.

respectively. Through the fringe patterns, we can easily illustrate the distribution of defective and non-defective instances. Thus, it is not hard to find that the unsupervised method obtains better performance in terms of NPMs at the cost of higher inspection efforts.

APPENDIX F
THE INFLUENCE OF CLASSIFIER ON EASC AND TUNEDMANUALUP

F.1 The Influence of Classifier on EASC

EASC, a supervised method proposed in this paper, builds a prediction model on the basis of state-of-the-art supervised classifiers and then resorts the testing instances for different practical usages. Therefore, in this section, to investigate the effect of the choice of EASC's underlying classifier, we make a comparison among six commonly used classifiers: Decision Tree, Random Forest, Logistic Regression, Naive Bayes, RBF Network and Support Vector Machine. Table 11 lists the introduction to all basic classifiers used in the paper. The first to third columns are the name of classifier, the abbreviation and brief description respectively. These classifier are widely used in CPDP scenario [7], [49]–[52]. Figure 2 and Figure 3 show the overall performance of EASC built with different classifiers on different datasets.

Figure 2 and Figure 3 present the value of EASC in terms of 11 performance measures when using six different basic

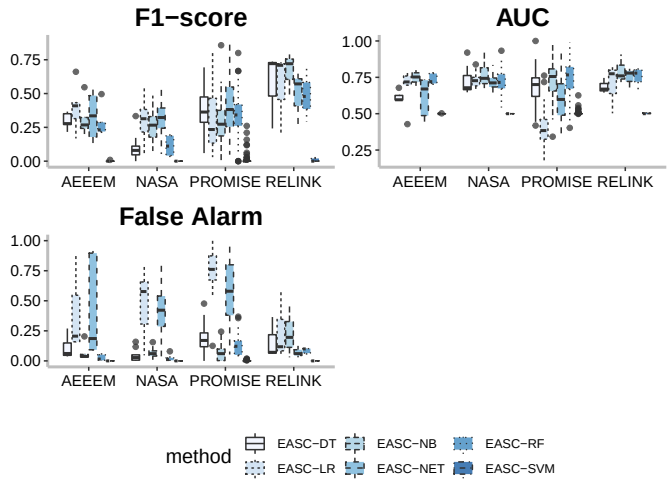


Fig. 2: Comparison of Different Classifiers in EASC in Terms of Non-Effort-Aware Performance Measures.

classifiers. Therefore, there are 11 sub-figures and each one illustrates the performance of EASC in terms of a specific performance measure. Besides, for every sub-figure, the x-axis represents different datasets while the y-axis represents the corresponding performance values. Notice since the

TABLE 6: Comparisons among supervised methods and **ManualDown** on four datasets in terms of **non-effort-aware performance measures**.

Measures	Datasets	CamargoCruz09-DT	Menzies11-RF	Turhan09-DT	Watanabe08-DT	ManualDown	ManualUp
$IFA^{\downarrow}$	AEEEM	1±1	0±0	2±5	1±1	1±2	30±417
	NASA	33±6341	28±5637(M)*	5±43(L)*	4±39	1±16	1268±7251939
	PROMISE	3±114(S)***	5±194(M)***	6±205(S)***	3±50(S)**	1±2	19±473
	RELINK	0±0	0±0	0±0	1±0	0±0	8±1
$PII@20\%^{\downarrow}$	AEEEM	0.22±0.00(L)**	0.22±0.00(L)**	0.22±0.00(L)**	0.22±0.00(L)**	0.02±0.00	0.69±0.00
	NASA	0.18±0.00(L)***	0.18±0.00(L)***	0.18±0.00(L)***	0.18±0.00(L)***	0.03±0.00	0.54±0.02
	PROMISE	0.22±0.00(L)***	0.22±0.00(L)***	0.22±0.00(L)***	0.22±0.00(L)***	0.03±0.00	0.68±0.01
	RELINK	0.17±0.00	0.17±0.00	0.17±0.00	0.17±0.00	0.04±0.00	0.68±0.00
$PII@1000^{\downarrow}$	AEEEM	0.02±0.00(L)**	0.02±0.00(L)**	0.02±0.00(L)**	0.02±0.00(L)**	0.00±0.00	0.19±0.02
	NASA	0.09±0.01(L)*	0.09±0.01(L)*	0.09±0.01(L)*	0.09±0.01(L)*	0.02±0.00	0.25±0.02
	PROMISE	0.08±0.02(L)***	0.08±0.02(L)***	0.08±0.02(L)***	0.08±0.02(L)***	0.03±0.00	0.35±0.04
	RELINK	0.08±0.00	0.08±0.00	0.08±0.00	0.08±0.00	0.02±0.00	0.48±0.05
$PII@2000^{\downarrow}$	AEEEM	0.02±0.00(L)**	0.02±0.00(L)**	0.02±0.00(L)**	0.02±0.00(L)**	0.00±0.00	0.26±0.02
	NASA	0.18±0.05	0.18±0.05	0.18±0.05	0.18±0.05	0.05±0.01	0.39±0.05
	PROMISE	0.14±0.05(L)***	0.14±0.05(L)***	0.14±0.05(L)***	0.14±0.05(L)***	0.06±0.03	0.45±0.05
	RELINK	0.18±0.04	0.18±0.04	0.18±0.04	0.18±0.04	0.05±0.00	0.61±0.07
$CostEffort@20\%^{\uparrow}$	AEEEM	**0.26±0.00(L)****	**0.20±0.03(L)***	0.24±0.01(L)**	**0.30±0.01(L)****	0.05±0.00	0.26±0.00
	NASA	0.07±0.01	0.09±0.00	0.13±0.01	0.11±0.02	0.10±0.00	0.18±0.02
	PROMISE	**0.29±0.02(L)*****	**0.27±0.02(L)*****	**0.28±0.02(L)*****	**0.26±0.01(L)*****	0.08±0.00	0.27±0.02
	RELINK	0.29±0.00	0.31±0.00	0.26±0.02	0.22±0.00	0.09±0.00	0.28±0.00
$CostEffort@1000^{\uparrow}$	AEEEM	**0.04±0.00(L)***	**0.05±0.00(L)***	**0.03±0.00**	**0.04±0.00(L)***	0.00±0.00	0.08±0.00
	NASA	0.06±0.00	0.06±0.00	0.08±0.01	0.05±0.00	0.05±0.00	0.08±0.02
	PROMISE	**0.09±0.02(S)***	0.06±0.01	0.06±0.01	**0.09±0.02(S)***	0.05±0.01	0.15±0.01
	RELINK	0.16±0.02	0.15±0.02	0.14±0.02	0.11±0.01	0.06±0.00	0.19±0.01
$CostEffort@2000^{\uparrow}$	AEEEM	**0.05±0.00(L)***	**0.07±0.00(L)***	**0.05±0.00(L)***	**0.04±0.00(L)***	0.01±0.00	0.12±0.01
	NASA	0.06±0.00	0.08±0.00	0.10±0.01	0.06±0.00	0.11±0.02	0.11±0.02
	PROMISE	**0.12±0.02(S)****	0.10±0.02	0.10±0.03	**0.13±0.03(S)****	0.08±0.02	0.18±0.02
	RELINK	0.21±0.05	0.29±0.10	0.24±0.09	0.29±0.12	0.12±0.03	0.24±0.00
$P_{opt}@20\%^{\uparrow}$	AEEEM	**0.49±0.01(L)***	0.49±0.02	0.40±0.00	**0.51±0.02(L)***	0.22±0.03	0.65±0.00
	NASA	0.34±0.04	0.36±0.04	0.34±0.04	0.35±0.03	0.41±0.04	0.49±0.04
	PROMISE	**0.45±0.04(L)*****	**0.39±0.03(L)*****	0.39±0.04(L)***	**0.43±0.05(L)*****	0.20±0.08	0.63±0.04
	RELINK	0.51±0.13	0.46±0.04	0.50±0.12	0.62±0.12	0.32±0.08	0.63±0.00

Notes: (1) *** means $p < 0.001$, ** means $p < 0.01$, * means $p < 0.05$.
(2) L/M/S: Large/Medium/Small effect size according to Cliff's delta.
(3) '↓' indicates 'the smaller the better'; '↑' indicates 'the larger the better'.

TABLE 7: The Distribution of Instance Inspection Effort and Instance Quality on AEEEM

Dataset	Project	Percentage of Defects					# Defect	Percentage of Efforts					# Total Effort	# Instances	Distribution
		10%	20%	30%	40%	50%		10%	20%	30%	40%	50%			Larger effort⟸⟹Lower effort
AEEEM	eclipse	0.35	0.54	0.68	0.76	0.79	206	0.59	0.74	0.83	0.89	0.93	224,055	997	
	equinox	0.21	0.36	0.53	0.63	0.73	129	0.55	0.72	0.83	0.90	0.95	39,534	324	
	lucene	0.20	0.39	0.50	0.58	0.67	64	0.58	0.73	0.82	0.88	0.92	73,184	691	
	mylyn	0.29	0.45	0.58	0.68	0.73	245	0.52	0.68	0.78	0.86	0.91	156,102	1,862	
	pde	0.25	0.46	0.59	0.70	0.78	209	0.42	0.60	0.72	0.81	0.87	146,952	1,497	

TABLE 8: The Distribution of Instance Inspection Effort and Instance Quality on NASA

Dataset	Project	Percentage of Defects					# Defect	Percentage of Efforts					# Total Effort	# Instances	Distribution
		10%	20%	30%	40%	50%		10%	20%	30%	40%	50%			Larger effort⟸⟹Lower effort
NASA	CM1	0.29	0.48	0.60	0.69	0.76	42	0.38	0.55	0.66	0.75	0.81	13,626	344	
	JM1	0.25	0.41	0.55	0.65	0.72	1,759	0.47	0.64	0.75	0.82	0.88	272,370	9,593	
	KC1	0.29	0.48	0.64	0.75	0.86	325	0.50	0.72	0.84	0.91	0.96	30,631	2,096	
	KC3	0.28	0.39	0.53	0.58	0.64	36	0.35	0.54	0.66	0.75	0.82	6,354	200	
	MC1	0.65	0.84	0.88	0.88	1.00	68	0.65	0.84	0.93	0.97	1.00	59,159	9,277	
	MC2	0.23	0.34	0.45	0.57	0.66	44	0.41	0.58	0.70	0.79	0.86	5,205	127	
	MW1	0.44	0.59	0.63	0.74	0.78	27	0.28	0.45	0.59	0.69	0.78	6,838	264	
	PC1	0.30	0.46	0.61	0.75	0.82	61	0.37	0.55	0.66	0.75	0.82	22,038	759	
	PC2	0.13	0.19	0.44	0.44	0.69	16	0.31	0.47	0.59	0.69	0.77	3,170	1,585	
	PC3	0.18	0.37	0.52	0.66	0.74	140	0.41	0.57	0.68	0.76	0.83	31,040	1,125	
	PC4	0.17	0.39	0.57	0.69	0.75	178	0.38	0.56	0.68	0.78	0.84	26,980	1,399	
	PC5	0.91	0.94	0.96	0.96	0.96	503	0.80	0.85	0.88	0.90	0.92	153,501	17,001	

TABLE 9: The Distribution of Instance Inspection Effort and Instance Quality on RELINK

Dataset	Project	Percentage of Defects					# Defect	Percentage of Efforts					# Total Effort	# Instances	Distribution
		10%	20%	30%	40%	50%		10%	20%	30%	40%	50%			Larger effort⟸⟹Lower effort
RELINK	Apache2.0	0.16	0.31	0.48	0.62	0.71	98	0.41	0.63	0.78	0.86	0.92	59,879	194	
	openintents	0.27	0.50	0.59	0.73	0.77	22	0.46	0.69	0.80	0.90	0.95	4,121	56	
	zxing1.6	0.16	0.28	0.43	0.57	0.68	118	0.47	0.68	0.79	0.86	0.91	12,811	399	

TABLE 10: The Distribution of Instance Inspection Effort and Instance Quality on PROMISE

Dataset	Project	Percentage of Defects					# Defect	Percentage of Efforts					# Total Effort	# Instances	Distribution
		10%	20%	30%	40%	50%		10%	20%	30%	40%	50%			Larger effort⟸⟹Lower effort
PROMISE	ant-1.3	0.40	0.55	0.65	0.80	0.90	20	0.36	0.55	0.70	0.79	0.87	37,699	125	
	ant-1.4	0.10	0.25	0.40	0.50	0.63	40	0.38	0.59	0.73	0.83	0.89	54,195	178	
	ant-1.5	0.34	0.59	0.69	0.72	0.81	32	0.45	0.65	0.77	0.85	0.90	87,047	293	
	ant-1.6	0.26	0.51	0.70	0.82	0.88	92	0.44	0.64	0.76	0.84	0.90	113,246	351	
	ant-1.7	0.33	0.57	0.69	0.78	0.87	166	0.45	0.64	0.75	0.84	0.90	208,653	745	
	arc	0.33	0.44	0.52	0.59	0.63	27	0.55	0.70	0.81	0.89	0.94	31,342	234	
	berek	0.31	0.56	0.81	0.94	1.00	16	0.64	0.75	0.84	0.91	0.94	32,320	43	
	camel-1.0	0.31	0.31	0.54	0.69	0.77	13	0.41	0.59	0.71	0.81	0.88	33,721	339	
	camel-1.2	0.14	0.24	0.35	0.45	0.54	216	0.41	0.61	0.75	0.84	0.90	66,302	608	
	camel-1.4	0.23	0.39	0.48	0.61	0.69	145	0.43	0.63	0.76	0.84	0.91	98,080	872	
	camel-1.6	0.19	0.31	0.43	0.51	0.62	188	0.45	0.64	0.76	0.85	0.91	113,055	965	
	ckjm	0.20	0.40	0.40	0.60	0.80	5	0.29	0.43	0.58	0.72	0.82	1,469	10	
	e-learning	0.60	0.60	1.00	1.00	1.00	5	0.42	0.59	0.72	0.80	0.87	3,639	64	
	forrest-0.7	0.20	0.60	0.60	0.60	0.80	5	0.26	0.45	0.61	0.74	0.83	4,930	29	
	ivy-1.1	0.14	0.29	0.44	0.59	0.67	63	0.55	0.72	0.81	0.88	0.93	27,292	111	
	ivy-1.4	0.44	0.56	0.69	0.75	0.88	16	0.58	0.75	0.84	0.90	0.94	59,286	241	
	ivy-2.0	0.45	0.65	0.75	0.83	0.88	40	0.52	0.73	0.83	0.90	0.94	87,769	352	
	jedit-3.2	0.20	0.42	0.53	0.68	0.78	90	0.63	0.75	0.83	0.89	0.93	128,883	272	
	jedit-4.0	0.25	0.48	0.67	0.72	0.75	75	0.62	0.76	0.83	0.89	0.93	144,803	306	
	jedit-4.1	0.28	0.51	0.63	0.73	0.81	79	0.61	0.74	0.83	0.89	0.93	153,087	312	
	jedit-4.2	0.38	0.60	0.73	0.83	0.92	48	0.56	0.70	0.80	0.87	0.92	170,683	367	
	jedit-4.3	0.36	0.55	0.55	0.64	0.64	11	0.54	0.69	0.80	0.88	0.93	202,363	492	
	kalkulator	0.17	0.33	0.33	0.33	0.83	6	0.32	0.50	0.64	0.71	0.82	4,022	27	
	log4j-1.0	0.29	0.47	0.62	0.76	0.79	34	0.38	0.57	0.70	0.80	0.87	21,549	135	
	log4j-1.1	0.22	0.43	0.59	0.73	0.84	37	0.35	0.53	0.68	0.78	0.86	19,938	109	
	log4j-1.2	0.11	0.20	0.30	0.39	0.50	189	0.40	0.58	0.71	0.80	0.88	38,191	205	
	lucene-2.0	0.19	0.34	0.46	0.57	0.67	91	0.47	0.63	0.76	0.84	0.90	50,596	195	
	lucene-2.2	0.13	0.26	0.39	0.47	0.57	144	0.48	0.65	0.77	0.85	0.91	63,571	247	
	lucene-2.4	0.14	0.27	0.38	0.49	0.58	203	0.52	0.69	0.80	0.87	0.92	102,859	340	
	nieruchomosci	0.30	0.60	0.70	0.80	0.90	10	0.35	0.56	0.70	0.78	0.87	4,754	27	
	pbeans1	0.15	0.30	0.40	0.50	0.60	20	0.76	0.87	0.90	0.95	0.97	5,572	26	
	pbeans2	0.30	0.50	0.60	0.60	0.80	10	0.73	0.83	0.90	0.95	0.98	15,125	51	
	pdftranslator	0.27	0.40	0.53	0.67	0.80	15	0.56	0.71	0.81	0.90	0.94	6,318	33	
	poi-1.5	0.12	0.28	0.40	0.53	0.63	141	0.40	0.57	0.70	0.79	0.86	55,428	237	
	poi-2.0	0.30	0.41	0.51	0.62	0.62	37	0.48	0.63	0.75	0.83	0.88	93,171	314	
	poi-2.5	0.13	0.25	0.38	0.47	0.57	248	0.49	0.64	0.75	0.83	0.89	119,731	385	
	poi-3.0	0.14	0.27	0.41	0.53	0.64	281	0.49	0.66	0.77	0.85	0.90	129,327	442	

continued on next page

TABLE 10: The Distribution of Instance Inspection Effort and Instance Quality on PROMISE. – continued from previous page

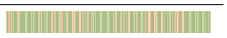
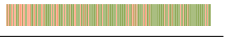
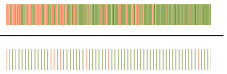
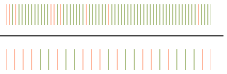
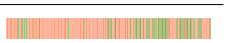
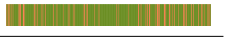
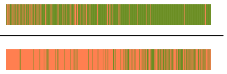
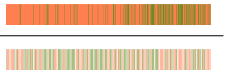
Dataset	Project	Percentage of Defects					# Defect	Percentage of Efforts					# Total Effort	# Instances	Distribution
		10%	20%	30%	40%	50%		10%	20%	30%	40%	50%			Larger effort⟺Lower effort
PROMISE	redaktor	0.11	0.15	0.30	0.44	0.48	27	0.34	0.51	0.64	0.74	0.83	59,280	176	
	serapion	0.33	0.67	0.67	0.67	0.78	9	0.53	0.67	0.78	0.83	0.88	10,505	45	
	skarbonka	0.22	0.33	0.44	0.78	0.89	9	0.43	0.62	0.78	0.87	0.94	15,029	45	
	sklebagd	0.17	0.33	0.50	0.58	0.67	12	0.39	0.54	0.65	0.74	0.83	9,602	20	
	synapse-1.0	0.44	0.69	0.88	0.88	0.88	16	0.32	0.52	0.65	0.75	0.84	28,806	157	
	synapse-1.1	0.22	0.38	0.50	0.60	0.68	60	0.35	0.55	0.69	0.78	0.86	42,302	222	
	synapse-1.2	0.21	0.42	0.57	0.65	0.73	86	0.36	0.58	0.70	0.80	0.87	53,500	256	
	systemdata	0.33	0.33	0.67	0.78	0.89	9	0.55	0.74	0.84	0.90	0.94	15,441	65	
	szybkafucha	0.21	0.29	0.36	0.43	0.57	14	0.35	0.49	0.65	0.74	0.85	1,910	25	
	termoproject	0.38	0.54	0.69	0.69	0.85	13	0.57	0.72	0.82	0.89	0.94	8,239	42	
	tomcat	0.39	0.65	0.75	0.82	0.87	77	0.54	0.74	0.84	0.91	0.95	300,674	858	
	velocity-1.4	0.10	0.18	0.27	0.38	0.47	147	0.62	0.75	0.84	0.90	0.94	51,713	196	
	velocity-1.5	0.11	0.24	0.39	0.52	0.64	142	0.63	0.76	0.84	0.90	0.95	53,141	214	
	velocity-1.6	0.13	0.32	0.47	0.60	0.68	78	0.62	0.76	0.85	0.91	0.95	57,012	229	
	workflow	0.20	0.30	0.45	0.55	0.65	20	0.46	0.61	0.72	0.81	0.87	4,125	39	
	wspomaganiepi	0.17	0.25	0.42	0.58	0.67	12	0.31	0.50	0.64	0.75	0.80	5,685	18	
	xalan-2.4	0.30	0.51	0.71	0.83	0.88	110	0.51	0.69	0.80	0.88	0.93	225,088	723	
	xalan-2.5	0.15	0.28	0.40	0.51	0.62	387	0.54	0.72	0.83	0.90	0.94	304,860	803	
	xalan-2.6	0.21	0.38	0.50	0.61	0.73	411	0.53	0.72	0.83	0.90	0.94	411,737	885	
	xalan-2.7	0.10	0.20	0.30	0.40	0.51	898	0.52	0.71	0.82	0.89	0.94	428,555	909	
	xerces-1.2	0.18	0.28	0.32	0.44	0.46	71	0.73	0.88	0.94	0.97	0.98	159,254	440	
	xerces-1.3	0.26	0.46	0.57	0.75	0.83	69	0.73	0.88	0.94	0.97	0.98	167,095	453	
	xerces-1.4	0.13	0.25	0.37	0.49	0.61	437	0.68	0.84	0.91	0.95	0.97	141,180	588	
	xerces-init	0.16	0.23	0.31	0.39	0.44	77	0.73	0.88	0.94	0.97	0.98	90,718	162	
	zuzel	0.23	0.46	0.62	0.69	0.85	13	0.38	0.65	0.82	0.89	0.94	14,421	29	

TABLE 11: The basic classifiers used in our experiment

Classifier	Abbr.	Brief Description
C4.5 Decision Tree [43]	DT	A tree structure consists of nodes and leafs. Each node of the tree represents a logical decision based on a single metric, while each leaf of the tree defines the classification. Information gain decides which attribute should be used for decision.
Random Forest [44]	RF	A random forest is composed of many random trees. Each random tree is a decision tree (e.g., C4.5).
Logistic Regression [45]	LR	A linear regression model estimates the likelihood of a classification with logistic function. For classification issues, logistic regression chooses the class with the highest likelihood.
Naive Bayes [46]	NB	The method estimates a score for each class based on a simplification of Bayes law.
RBF Network [47]	NET	One of the types of artificial neural network with Radial Basis Functions (RBFs) as neurons.
Support Vector Machine [48]	SVM	It determines a hyperplane that separates the positive from the negative samples. The hyperplane is determined in the kernel space of the data, i.e., a transformation of the data in a higher dimensional space using a kernel function. We use RBFs as kernel function in this paper.

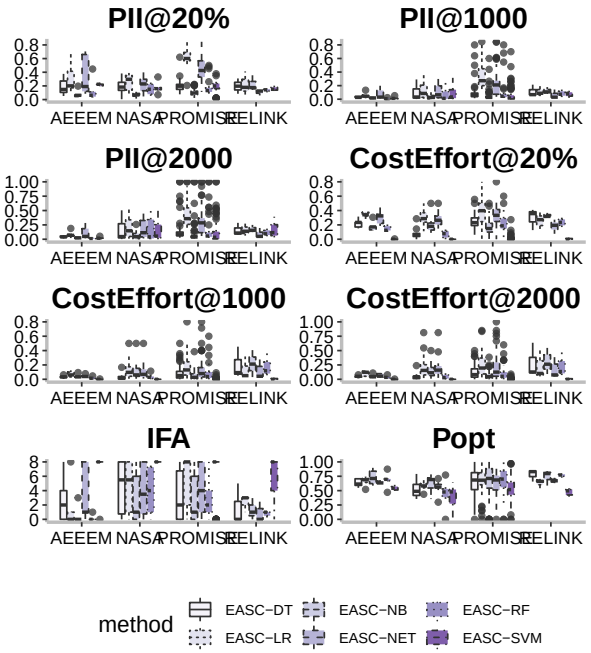


Fig. 3: Comparison of Different Classifiers in EASC in Terms of Effort-Aware Performance Measures.

large range of *IFA* value (i.e., **0∼17001**), we truncate those values which are large than 22 and make them equal to 22 for better illustration presentation. We choose 22 as the cut-point since the number of values which are larger than 22 occupies a small proportion (i.e., 121/492) and the number 22 is statistically the upper quartile value for *IFA*.

As shown in Figure 2 and Figure 3, when EASC built on different basic classifiers, the performances of EASC appear similarly in terms of different performance measures. Through observing this figure, we can obtain the following two conclusions:

- the six state-of-the-art classifiers have similar prediction ability since they can achieve similar performance in terms of NPMs and EPMs;
- EASC does not rely on the classifiers used, at least on the four datasets. However, through further analysis of the results, we find that EASC using Naive Bayes wins more times than any other classifiers considering the average performances.

We list the total numbers of wins and ties for each classifier on different projects in Table 12. As shown in Table 12, the first column represents classifiers, the following 11 columns represent different performance measures used in this paper and the last column lists the sum of each classifier. We sum up the number of cases that EASC built on a special classifier obtains the best performance or obtains the same performance with others which obtain the best performance. At last, we sort classifiers in descending order according to total win/tied numbers and find that NB ranks first. Therefore, we use NB as the default basic classifier in EASC.

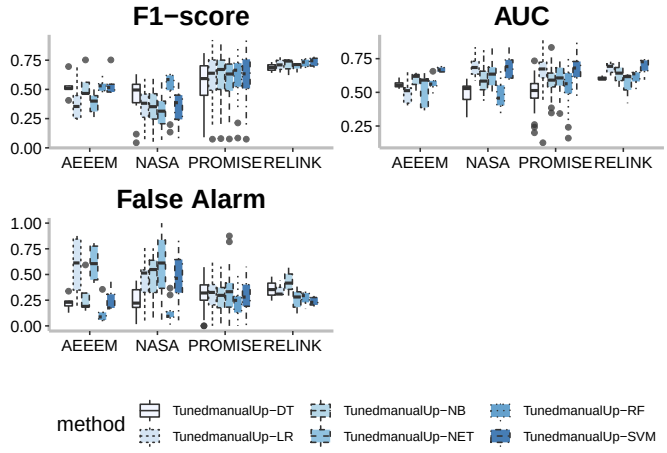


Fig. 4: Comparison of Different Classifiers in $TunedmanualUp$ in Terms of Non-effort-aware Performance Measures.

EASC is insensitive to basic classifiers and can obtain stable performances on various projects using different classifiers.

F.2 The Influence of Classifier on TunedmanualUp

Menzies et al. [53] found that manualUp tuned with a defect predictor could achieve better performance. In particular, in the phase of model building, a defect predictor should be trained on training instances. In the phase of model applying, the defect predictor firstly makes a binary decision (e.g., defective or clean) on testing instances. Then, all instances identified as defective are sorted in ascending order by LOC. Therefore, the choice of $TunedmanualUp$ underlying classifier can infect the sorting order of testing instances.

Therefore, in this section, to investigate the effect of the choice of $TunuedmanualUp$'s underlying classifier, we make a comparison among six commonly used classifiers: Decision Tree, Random Forest, Logistic Regression, Naive Bayes, RBF Network and Support Vector Machine. The introduction to all basic classifiers can be found in Table 11.

Figure 4 and Figure 5 present the overall performance of $TunedmanualUp$ in terms of 11 performance measures when using six different basic classifiers. Notice since the large range of *IFA* value (i.e., 0∼435), we truncate those values which are large than 8 and make them equal to 8 for better illustration presentation. We choose 8 as the cut-point since the number of values which are larger than 8 occupies a small proportion (i.e., 109/492) and the number 8 is statistically the upper quartile value for *IFA*.

As shown in Figure 4 and Figure 5, when $TunedmanualUp$ built on different basic classifiers, $TunedmanualUp$ achieves similar performances in terms of different performance measures. Through observing this figure, we can find that the six state-of-the-art classifiers have similar prediction ability since they can achieve similar performance in terms of NPMs and EPMs.

TABLE 12: The Win/Tie Numbers of Different Classifiers in EASC on Four Datasets

Classifiers	F1-score	AUC	False Alarm	IFA	PII			CostEffort			Popt	Total
					20%	1000	2000	20%	1000	2000		
EASC-NB	18	32	16	41	56	27	33	10	11	8	30	282
EASC-SVM	1	2	82	14	7	44	39	0	1	1	5	196
EASC-LR	14	0	0	16	0	0	2	40	52	53	14	191
EASC-RF	8	37	7	29	15	13	13	10	9	9	12	162
EASC-NET	24	4	0	21	4	2	5	28	30	31	9	158
EASC-DT	18	8	5	32	3	1	4	11	13	14	16	125

TABLE 13: The Win/Tie Numbers of Different Classifiers in $TunedmaunalUp$ on Four Datasets

Classifiers	F1-score	AUC	False Alarm	IFA	PII			CostEffort			Popt	Total
					20%	1000	2000	20%	1000	2000		
TunedmanualUp-NB	37	4	58	27	60	28	35	15	28	22	32	346
TunedmanualUp-SVM	18	42	10	52	14	29	28	20	22	19	15	269
TunedmanualUp-LR	15	21	6	40	3	15	10	20	31	28	16	205
TunedmanualUp-RF	13	6	10	38	17	29	27	14	24	14	11	203
TunedmanualUp-NET	8	8	11	29	12	20	12	14	21	18	9	162
TunedmanualUp-DT	8	3	8	19	3	1	3	22	30	30	18	145

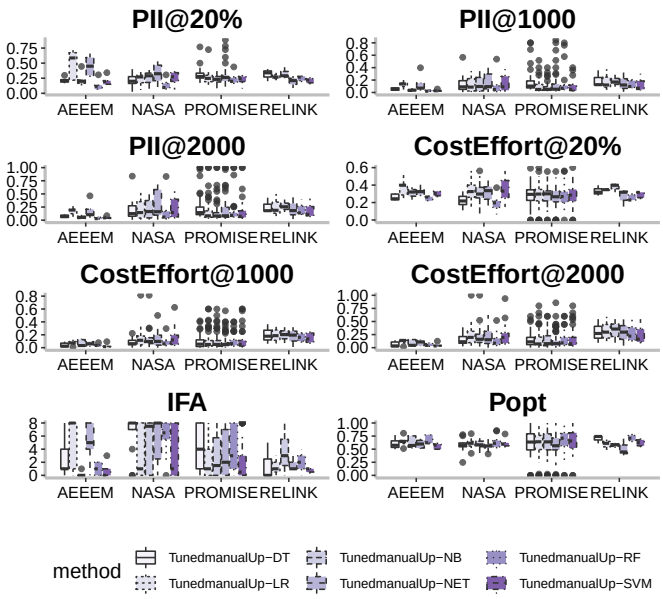


Fig. 5: Comparison of Different Classifiers in $TunedmanualUp$ in Terms of Effort-Aware Performance Measures.

We list the total numbers of wins and ties for each classifier on different projects in Table 13. As shown in Table 13, we find that $TunedmanualUp$ using Naive Bayes wins more times than any other classifiers considering the average performances. Therefore, we use NB as the default basic classifier in $TunedmanualUp$.

APPENDIX G
THE CORRELATION BETWEEN PERFORMANCE MEASURES.

In this paper, we totally consider 11 performance measures, which can be divided into two groups: non-effort-aware performance measures (NPMs) and effort-aware performance measures (EPMs). In particular, as for NPMs, three performance measures are considered which are widely used in the scenario of software defect prediction. As for EPMs, eight performance measures from four different types are considered which are most recently proposed and are not considered in Zhou et al.'s work.

Different software modules may have different sizes. That is, the lines of code in software modules may vary from hundreds of LOC to thousands of LOC. Therefore, to comprehensively investigate $PII@L$ and $CostEffort@L$, two kinds of $PII@L$ and $CostEffort@L$ are considered: 1) relative LOC of PII and CostEffort (e.g., 20%); 2) absolute LOC of PII and CostEffort (e.g., 1000, 2000). These performance measures consider costs from different perspectives: inspection costs, revenue costs, and impact on developers.

We make a correlation analysis among the all performance measures on all projects and all methods. The results are shown in Figure 6:

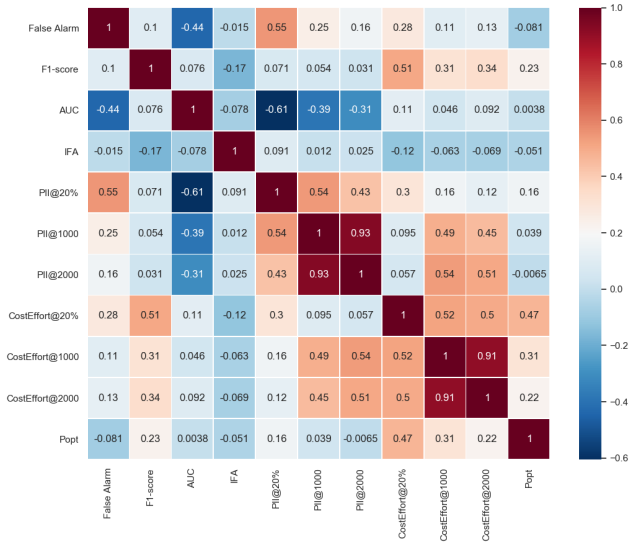


Fig. 6: Correlation analysis among all performance measures on all projects and all methods.

In the above figure, different colors indicate different degrees of correlation between two performance measures. From this figure, we make the following observations: (1) $PII@1000$ ($CosfEffort@1000$) has a high correlation with

PII@2000 (*CosfEffot@2000*), which is also obvious since they has same definition with different inspection effort. (2) *F1-score* has a 51% correlation with *CostEffort@20%*. The reason behinds this observation is that *CostEffort@L* has the same meaning with *Recall* and *CostEffort@L* can be treated as the effort-aware version of *Recall*. Besides, *F1-score* is highly related to *Recall*. (3) *PII@20%* has a 55% correlation with *False Alarm*. It is obvious that if there are many false alarm in modeling application, then it requires developers to switch the context more frequently. Therefore, *False Alarm* has a relationship with *PII@20%*. (4) For other performance measures, there is not obvious correlation between two performance measures.

According to above analysis, in this revision, we keep *PII@1000* and *PII@2000* since they are the absolute version of *PII@L*. We also keep *CostEffort@L* for the same reason. Besides, we also keep *F1-score*, *CostEffort@20%*, *PII@20%* and *False Alarm* for two reasons: 1) the correlation between each pair of performance measures is not large; 2) these performance measures are proposed for different goals (i.e., effort-aware and non-effort-aware).

REFERENCES

[1] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," IEEE Trans. Software Eng., vol. 44, no. 9, pp. 811–833, 2018.

[2] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," IEEE Transactions on Software Engineering, vol. 42, no. 10, pp. 977–998, 2016.

[3] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," Empirical Software Engineering, vol. 21, no. 1, pp. 43–71, 2016.

[4] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 409–418.

[5] D. Ryu and J. Baik, "Effective multi-objective naïve bayes learning for cross-project defect prediction," Applied Soft Computing, vol. 49, pp. 1062–1077, 2016.

[6] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," Information and Software Technology, vol. 54, no. 3, pp. 248–256, 2012.

[7] X. Chen, D. Zhang, Y. Zhao, Z. Cui, and C. Ni, "Software defect number prediction: Unsupervised vs supervised methods," Information and Software Technology, vol. 106, pp. 161–181, 2019.

[8] Z. Xu, S. Li, Y. Tang, X. Luo, T. Zhang, J. Liu, and J. Xu, "Cross version defect prediction with representative data via sparse subset selection," in Proceedings of the 26th Conference on Program Comprehension, ser. ICPC '18. New York, NY, USA: ACM, 2018, pp. 132–143. [Online]. Available: http://doi.acm.org/10.1145/3196321.3196331

[9] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," Information and Software Technology, 2018.

[10] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Combined classifier for cross-project defect prediction: an extended empirical study," Frontiers of Computer Science, vol. 12, no. 2, pp. 280–296, 2018.

[11] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018, pp. 209–220.

[12] Y. Liu, Y. Li, J. Guo, Y. Zhou, and B. Xu, "Connecting software metrics across versions to predict defects," in 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018, pp. 232–243.

[13] S. Hosseini, B. Turhan, and 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, vol. 95, pp. 296–312, 2018.

[14] A. Boucher and M. Badri, "Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison," Information and Software Technology, vol. 96, pp. 38–67, 2018.

[15] B. Turhan, A. T. Mısırlı, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," Information and Software Technology, vol. 55, no. 6, pp. 1101–1118, 2013.

[16] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014, pp. 172–181.

[17] C. Ni, W. Liu, Q. Gu, X. Chen, and D. Chen, "Fesch: A feature selection method using clusters of hybrid-data for cross-project defect prediction," in Proceedings of the 41st Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2017, pp. 51–56.

[18] S. Amasaki, "Cross-version defect prediction using cross-project defect prediction approaches: Does it work?" in Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering. ACM, 2018, pp. 32–41.

[19] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," Information and Software Technology, vol. 100, pp. 87–102, 2018.

[20] Z. Li, X.-Y. Jing, X. Zhu, and H. Zhang, "Heterogeneous defect prediction through multiple kernel learning and ensemble learning," in Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017, pp. 91–102.

[21] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," Software Quality Journal, vol. 25, no. 1, pp. 235–272, 2017.

[22] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, and S. Ying, "Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction," Automated Software Engineering, vol. 25, no. 2, pp. 201–245, 2018.

[23] T. Mende and R. Koschke, "Effort-aware defect prediction models," in Proceedings of the 14th European Conference on Software Maintenance and Reengineering. IEEE, 2010, pp. 107–116.

[24] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, 2016, pp. 165–176.

[25] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in Proceedings of the international symposium on software testing and analysis. ACM, 2011, pp. 199–209.

[26] J. H. Feng Wang and Y. Ma, "A top-k learning to rank approach to cross-project software defect prediction," in Proceedings of 25th Asia-Pacific Software Engineering Conference, 2018.

[27] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," IEEE Transactions on Reliability, vol. 67, no. 3, pp. 885–896, 2018.

[28] W. N. Poon, K. E. Bennin, J. Huang, P. Phannachitta, and J. W. Keung, "Cross-project defect prediction using a credibility theory based naive bayes classifier," in Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on. IEEE, 2017, pp. 434–441.

[29] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "On the multiple sources and privacy preservation issues for heterogeneous defect prediction," IEEE Transactions on Software Engineering (TSE), 2017.

[30] Y. Fan, C. Lv, X. Zhang, G. Zhou, and Y. Zhou, "The utility challenge of privacy-preserving data-sharing in cross-company defect prediction: An empirical study of the cliff&morph algorithm," in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017, pp. 80–90.

[31] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, and B. Xu, "How far we have progressed in the journey? an examination of cross-project defect prediction," ACM Trans. Softw. Eng. Methodol., vol. 27, no. 1, pp. 1:1–1:51, 2018.

[32] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," IEEE Transactions on Software Engineering, vol. 39, no. 9, pp. 1208–1215, 2013.

[33] D. Gray, D. Bowes, N. Davey, and Y. Sun, "The misuse of the nasa metrics data program data sets for automated software

defect prediction," in Proceedings of Evaluation & Assessment in Software Engineering, 2011, pp. 96–103.
[34] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in Proceedings of the 7th Mining Software Repositories (MSR). IEEE, 2010, pp. 31–41.
[35] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011, pp. 15–25.
[36] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceeding of International Conference on Predictive MODELS in Software Engineering, 2010, pp. 1–10.
[37] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," 2012.
[38] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," IEEE Transactions on Software Engineering, vol. 45, pp. 683–711, 2019.
[39] A. E. Camargo Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, 2009, pp. 460–463.
[40] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," Empirical Software Engineering, vol. 14, no. 5, pp. 540–578, 2009.
[41] T. Menzies, A. Butcher, A. Marcus, and D. Zimmermann, Thomas a nd Cok, "Local vs. global models for effort estimation and defect prediction," in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011, pp. 343–351.
[42] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in Proceedings of the 4th international workshop on Predictor models in software engineering. ACM, 2008, pp. 19–24.
[43] J. R. Quinlan, C4.5: programs for machine learning. Elsevier, 2014.
[44] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.
[45] D. R. Cox, "The regression analysis of binary sequences," Journal of the Royal Statistical Society. Series B (Methodological), pp. 215–242, 1958.
[46] S. J. Russell and P. Norvig, Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,, 2016.
[47] D. S. Broomhead and D. Lowe, "Radial basis functions, multivariable functional interpolation and adaptive networks," Royal Signals and Radar Establishment Malvern (United Kingdom), Tech. Rep., 1988.
[48] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," IEEE Intelligent Systems and their applications, vol. 13, no. 4, pp. 18–28, 1998.
[49] X. Xia, D. Lo, X. Wang, and X. Yang, "Collective personalized change classification with multiobjective search," IEEE Transactions on Reliability, vol. 65, no. 4, pp. 1810–1829, 2016.
[50] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," Journal of Computer Science and Technology, vol. 32, no. 6, pp. 1090–1107, 2017.
[51] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in Proceedings of the 23rd international conference on Machine learning. ACM, 2006, pp. 161–168.
[52] T. Van Gestel, J. A. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle, "Benchmarking least squares support vector machine classifiers," Machine learning, vol. 54, no. 1, pp. 5–32, 2004.
[53] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," Automated Software Engineering, vol. 17, no. 4, pp. 375–407, 2010.