Supplementary material for:
An extensive study on inconsistent labels in multi-version-project defect data sets

Shiran Liu[1,2] Zhaoqiang Guo[1,2] Yanhui Li[1,2]* Chuanqi Wang[1,2]
Lin Chen[1,2] Zhongbin Sun[3,4] Yuming Zhou[1,2]* Baowen Xu[1,2]

[1]State Key Laboratory for Novel Software Technology, Nanjing University
[2]Department of Computer Science and Technology, Nanjing University
[3]Mine Digitization Engineering Research Center of Ministry of Education, Xuzhou 221116, China
[4]School of Computer Science & Technology, China University of Mining and Technology

*Corresponding author

Appendix A. The running time of our TSILI algorithm

The method we adopted to generate the source code databases required by the TSILI algorithm is to download the codes corresponding to each version from the official website of each target project, and then use the Understand[1] tool to parse the code to generate source code databases (.udb file). We write a Python[2] script to implement the TSILI algorithm. In the second stage of TSILI, the source code of each module is parsed and filtered (b9 in Fig. 17) based on the Python API (application programming interface) provided by the Understand tool.

In order to observe the time required for the TSILI algorithm to detect inconsistent labels on projects of different orders of magnitude, we selected Log4j, Hive, and Eclipse projects from multi-version-project defect data sets Metrics-Repo-2010 [1], JIRA-RA-2019 [2], and ECLIPSE-2007 [3], respectively, and then ran TSILI and recorded the time spent. Table 1 lists the details of these three projects and the single thread running time of TSILI. The 2nd column lists the versions included in each project. The 3rd column lists the order of magnitude of the project size, where *n*, *totalIns*, and *sumSLOC* represent the number of versions, the total number of instances of all versions, and the total number of code lines of all versions, respectively. The 4th column reports the running time of TSILI. These three projects (Log4j, Hive, and Eclipse) were selected because they represented orders of magnitude of the size of the projects in their respective data sets (the ECLIPSE-2007 data set only has the Eclipse project). In addition, the size of the total number of instances of these three projects varies in turn by one order of magnitude, which is conducive to observe the running time of TSILI under different orders of magnitude data sets.

Table 1 shows that the running time of TSILI is positively correlated with the size of the data set. For the project with hundreds of instances (i.e., Log4j project), the running time is at the second level. For the projects with ten thousands of instances (i.e., Eclipse project), the running time is at the hour level. Because TSILI is an offline algorithm, the hour-level (even minute-level or second-level) running time is acceptable in practice.

Table 1. Time consuming of the TSILI algorithm

Dataset	Project (versions)	n, totalIns, sumSLOC	Running Time	Experimental environment
Metrics-Repo-2010	Log4j (1.0, 1.1, 1.2)	n=3, totalIns=411, sumSLOC=74857	≈ 25 seconds	Inter(R) Core(TM) i7-7700 CPU @ 3.6GHz and 16G RAM
JIRA-RA-2019	Hive (0.9.0, 0.10.0, 0.12.0)	n=3, totalIns=5285, sumSLOC=974774	≈ 11 minutes	
ECLIPSE-2007	Eclipse (2.0, 2.1, 3.0)	n=3, totalIns=25203,	≈ 3 hours	

[1] https://scitools.com
[2] https://www.python.org

		sumSLOC=3089619		

Appendix B. Software metrics used in the MA-SZZ-2020 and filtered IND-JLMIV+R-2020 data sets

As mentioned in Section 6.2 of this paper, there are some versions in the IND-JLMIV+R-2020 data set [4] that are not suitable for model training or prediction. Therefore, before the experiments of RQ2 and RQ3, we filtered out the inappropriate versions and kept only 33 versions. Table 2 shows the version information of our filtered IND-JLMIV+R-2020 data set.

Table 2. The IND-JLMIV+R-2020 defect data sets used in our study

Dataset	Project	Versions	#Instances (#IL-instances)	%Defective	#Metrics
Used in RQ1: IND-JLMIV+R-2020 (semi-automatic SZZ-based)	38 projects	395 versions	3~1708 (0~35)	0~36%	4198 (44)
Filtered for RQ2 and RQ3: IND-JLMIV+R-2020 (semi-automatic SZZ-based)	Calcite	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11	1075~1331 (13~26)	6~17%	44
	Knox	0.3.0, 0.4.0, 0.5.0, 0.6.0, 0.7.0, 0.8.0	388~594 (10~35)	5~13%	
	Kylin	1.5.0, 2.0.0, 2.1.0, 2.2.0	739~1003 (10~18)	4~8%	
	Mahout	0.4, 0.5	763~777 (11~16)	7~10%	
	Manifoldcf	2.4, 2.5, 2.6, 2.7, 2.8	1020~1041 (10~11)	1~3%	
	Nutch	1.10, 2.0, 2.1, 2.2	252~500 (10~12)	5~13%	

Table 3 describes the size, complexity, coupling, and inheritance metrics in the MA-SZZ-2020 we collected and the filtered IND-JLMIV+R-2020 data set. In Table 3, column "Type" represents the type to which each metric belongs to, column "Name" gives the acronym of each metric, column "Definition" provides an informal description of the corresponding metric, and column "Tool for measuring metrics" gives the source of the tool that we measure metrics from. Note that inheritance metrics are indeed a form of coupling metrics. In practice, however, many researchers distinguish inheritance metrics from coupling metrics. Our study follows a metric classification framework similar to that in Briand et al. [5].

Table 3. List of metrics in the MA-SZZ-2020 data set

Type	Name	Definition	Tool for measuring metrics
Size Metrics	SLOC (loc in data set)	the non-commentary source lines of code in a class	We used the Perl script developed in previous studies [6, 7] to collect metrics based on the udb database, where the udb database is generated by the commercial software Understand.
	NMIMP	the number of methods implemented in a class	
	NumPara	sum of the number of parameters of the methods implemented in a class	
	NM	the number of methods in a class, both inherited and non-inherited	
	NAIMP	the number of attributes in a class excluding inherited ones	
	NA	the number of attributes in a class including both inherited and non-inherited	
	Stms	the number of declaration and executable statements in the methods of a class	
	Nmpub	number of public methods implemented in a class	
	NMNpub	number of non-public methods implemented in a class	
	NIM	Number of Instance Methods	
	NCM	Number of Class Methods	
	NLM	Number of Local Methods	
	AvgSLOC	Average Source Lines of Code	
Complexity Metrics	CDE	Class Definition Entropy	
	CIE	Class Implementation Entropy	
	WMC	Weighted Method Per Class	
	SDMC	Standard Deviation Method Complexity	
	AvgWMC	Average Weight Method Complexity	
	CCMax	Maximum cyclomatic complexity of a single method of a class	
	NTM	Number of Trivial Methods	
Coupling Metrics	CBO	Coupling Between Object	
	DAC	Data Abstraction Coupling: Type is the number of attributes of other classes.	
	DACquote	Data Abstraction Coupling: Type is the number of other classes.	
	ICP	Information-flow-based Coupling	
	IHICP	Information-flow-based inheritance Coupling	
	NIHICP	Information-flow-based non-inheritance Coupling	
Inheritance Metrics	NOC	Number Of Child Classes	
	NOP	Number Of Parent Classes	
	DIT	Depth of Inheritance Tree	
	AID	Average Inheritance Depth of a class	

	CLD	Class-to-Leaf Depth	
	NOD	Number Of Descendants	
	NOA	Number Of Ancestors	
	NMO	Number of Methods Overridden	
	NMI	Number of Methods Inherited	
	NMA	Number Of Methods Added	
	SIX	Specialization IndeX = NMO * DIT / (NMO + NMA + NMI)	
	PII	Pure Inheritance Index.	
	SPA	static polymorphism in ancestors	
	SPD	static polymorphism in descendants	
	DPA	dynamic polymorphism in ancestors	
	DPD	dynamic polymorphism in descendants	
	SP	static polymorphism in inheritance relations	
	DP	dynamic polymorphism in inheritance relations	

Appendix C. Detailed data analysis method for RQ2 (influence analysis on prediction performance)

Prediction performance evaluation. We evaluate the prediction performance of a defect prediction model under two scenarios: classification and ranking. For a given test data set, assume that: (1) n_0 is the number of clean instances; (2) n_1 is the number of defective instances; (3) N is the total number of instances; and (4) q is the actual defect percentage. Then, we have $N = n_0 + n_1$ and $q = n_1/N$. For each instance in this test set, a defect prediction model can output a probability of being defective. In the classification scenario, a threshold (0.5 in default) is chosen to classify an instance as defective if the predicted probability is larger than the threshold and otherwise not defective. Consequently, there are four outcomes: *TP* (the set of modules correctly classified as defective), *TN* (the set of modules correctly classified as not defective), *FP* (the set of modules incorrectly classified as defective), and *FN* (the set of modules incorrectly classified as not defective). Clearly, $N = |TP| + |FP| + |TN| + |FN|$, and $q = (|TP| + |FP|) / N$. In our study, we use the following indicators to evaluate the classification performance of a RF model.

- Matthews Correlation Coefficients (*MCC*) [8-10]. *MCC* measures a correlation coefficients between actual and predicted outcomes using the following calculation: $\frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$. An *MCC* value ranges from -1 to +1, where an *MCC* value of 1 indicates a perfect prediction, and -1 indicates total disagreement between the prediction.
- F_1 [11]: the harmonic mean of precision (i.e. $p = |TP| / (|TP|+|FP|)$) and recall (i.e. $r = |TP| / (|TP|+|FN|)$), i.e. $2 \times p \times r / (p + r)$.
- *AUC* [12]: the area under ROC (Receiver Operating Characteristic) curve, which is defined on a 2-dimensional plot in which the *x*-axis is TPR (i.e. $|TP|/(|TP|+|FN|)$) and the *y*-axis is the FPR (i.e. $|FP|/(|TN|+|FP|)$).
- *ER* (Effort Reduction) [13]: the proportion of the reduced number of modules to be inspected (i.e., effort) compared with a random model that achieves the same recall. According to the prediction model, $|TP|+|FP|$ modules will be inspected. According to a random model that achieves the same recall, $|TP|/(|TP|+|FN|) \times N$ modules will be inspected. Therefore, the reduced effort is:

$$ER = \frac{|TP|/(|TP|+|FN|)\times N-(|TP|+|FP|)}{|TP|/(|TP|+|FN|)\times N} = 1 - \frac{q}{p}$$

- *RI* (Recall Increase) [13]: the proportion of the increased recall compared with a random model when inspecting the same number of instances (i.e., the same effort). According to a random model, inspecting $|TP|+|FP|$ modules will lead to a recall of $(|TP|+|FP|)/N$. Therefore, the recall increase is:

$$RI = \frac{r-(|TP|+|FP|)/N}{(|TP|+|FP|)/N} = \frac{r}{q} - 1$$

Of the above five indicators, *MCC*, F_1 and *AUC* are three popular classification performance indicators in the

literature. However, they are non-effort-aware. In contrast, *ER* and *RI* are two effort-aware indicators. For each indicator, a larger value means a better performance.

In the ranking scenario, the instances in the test data set are ranked in descending order according to their predicted probability of being defective. In our study, we use the following indicators to evaluate the ranking performance of a RF model.

- *AP* (Average Precision): the average precision of defective instances. Let $p(k)$ be the precision calculated by considering only the ranked instances from rank 1 through k. Assume that $rel(k)$ indicates if the kth instance is defective ($rel(k) = 1$) or not ($rel(k) = 0$). Then, $AP = \frac{\sum_{k=1}^{N} p(k) \times rel(k)}{n_1}$.
- *RR* (Reciprocal Rank): the reciprocal of the rank of the first real defective instance.
- P_{opt} [14-16]: the area under the cost-effectiveness curve normalized to the optimal and worst models in an Alberg diagraph. In such a diagraph, the cumulative percentage of SLOC of the top modules selected from the instance ranking (the x-axis) is plotted against the cumulative percentage of defects found (the y-axis). Formally, for a model m, P_{opt} can be defined as:

  $$P_{opt} = 1 - \frac{Area(optimal) - Area(m)}{Area(optimal) - Area(worst)}$$

  Here, $Area(m)$, $Area(optimal)$ and $Area(worst)$ represent the area under the curve corresponding to the prediction model, the optimal model, and the worst model, respectively. In the optimal model and the worst model, instances are, respectively, ranked in decreasing and ascending order according to their actual defect density.
- *ACC* [14]: the recall of defective instances when using 20% of the entire effort required to inspect all instances (i.e. 20% of the total SLOC) to inspect the top ranked instances.

Of the above four indicators, *AP* and *RR* are two non-effort-aware indicators, in which the order of defective instances in a ranking is considered. They are originally from the field information retrieval [17] but appropriate for evaluating the ranking performance of a defect prediction model. In particular, $RR = 1 / (IFA + 1)$, where *IFA* is a recently proposed indicator [18] to characterize the number of Initial False Alarms encountered before the first defective instance is found in a rank. In contrast, P_{opt} and *ACC* are two popular effort-aware indicators, in which the module size corresponding to an instance is used as the proxy of the effort to inspect the instance. For each indicator, a larger value means a better performance.

Statistical performance comparison. Let $perf(m_1)$ be the performance of model m_1 and $perf(m_2)$ be the performance of model m_2. In our context, the performance measure *perf* can be *MCC* or *ACC*. In particular, when investigating the influence of inconsistent labels in a training set, m_1 is NC and m_2 is CC; when investigating the influence of inconsistent labels in a test set, m_1 is NN and m_2 is NC. For a pair of training and test set, a nature idea is to use the absolute value of *diff* to evaluate the influence of inconsistent labels [19]:

$$diff(m_1, m_2) = \frac{perf(m_1) - perf(m_2)}{perf(m_2)} \times 100\%$$

As can be seen, a positive *diff* means that inconsistent labels lead to overestimation of the prediction model performance (optimistic estimation), while a negative *diff* means that inconsistent labels lead to underestimation of the prediction model performance (conservative estimation). No matter whether the *diff* is positive or negative, it is clear that the smaller the absolute value of *diff* is, the less the influence is. However, the above analysis only considers the absolute performance *perf*. In practice, for a given test set, it is easy to use a random model *random* to predict defect-proneness. The random model can be regarded as the description of the difficulty of the problem

itself. For practitioners, the premise of using a model is that it should have a higher performance than *random*. In this sense, it is more important for practitioners to evaluate *perfGain*, the relative performance of a model with respect to *random* [20]. In our context, *perfGain*(m_1) = *perf*(m_1) − *perf*(*random*) and *perfGain*(m_2) = *perf*(m_2) − *perf*(*random*). Therefore, in this study, we also employ the absolute value of *pgr* (performance gain ratio) to evaluate the influence of inconsistent labels in *S* on prediction performance:

$$pgr(m_1, m_2) = \frac{perfGain(m_1) - perfGain(m_2)}{perfGain(m_2)} \times 100\%$$

The advantage of using *pgr* (performance gain ratio) indicator [20] is that the influence of the difficulty of the problem itself is considered. By eliminating the impact of the difficulty of the problem itself, it will be more fair and meaningful to observe or compare the performance difference between the two models. Although the *pgr* indicator itself has not been widely used in the existing literatures, the indicator using similar ideas (subtracting the difficulty of the problem itself, i.e., subtracting the performance of the random model), such as CE indicator, has been widely adopted in the existing literatures [21, 22]. In summary, *diff*(NC, CC) and *pgr*(NC, CC) are used for investigating the influence of inconsistent labels in the training data, while *diff*(NN, NC) and *pgr*(NN, NC) are used for investigating the influence of inconsistent labels in the test data.

Assume that a test set consists of *N* instances, in which n_1 are defective. For a random model *random*, an instance will have an equal probability (i.e. 0.5) to be predicted as clean or defective. Consequently, we can conclude that:

- TP = $N/2 * n_1/N = n_1/2$
- FP = $N/2 - n_1/2 = (N-n_1)/2$
- TN = $N/2 * (1-n_1/N) = (N-n_1)/2$
- FN = $N/2 - (N-n_1)/2 = n_1/2$

Therefore, we calculate *MCC*, F_1, *AUC*, *ER*, *RI*, P_{opt}, *ACC*, *AP*, *RR*, as:

- $MCC = 0$
- $F_1(random) = \frac{2 \times n_1/N \times 0.5}{n_1/N + 0.5}$
- $AUC(random) = 0.5$
- $ER(random) = 0$
- $RI(random) = 0$
- $AP(random) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{i}\left(\frac{C_{i-1}^{k-1} \times C_{N-i}^{n_1-k}}{C_N^{n_1}} \times \frac{k}{i}\right)$
- $RR(random) = \sum_{i=1}^{N-n_1+1}\left(\frac{1}{i} \times \frac{C_{N-i}^{n_1-1}}{C_N^{n_1}}\right)$
- $P_{opt}(random) = 0.5$
- $ACC(random) = 0.2$

The reasoning process of *AP* formula. We consider the calculation of *AP* as the sum of contribution of each position *i* ($1 \leq i \leq N$) with defective modules. For each *i*, assuming that there are defective modules at the current position *i*, the remaining n_1-1 defective modules need to be placed on both sides of position *i*, that is, one part of the defective modules should be placed on 1 to *i*-1, and the other part should be placed on *i*+1 to *N*. At this time, (1) the probability that there is a defective module at the current position *i* is $\frac{C_{N-1}^{n_1-1}}{C_N^{n_1}}$; (2) the probability that the defective module at the current position *i* is the *k*th defective module (i.e., *k*-1 ($k \leq i$) defective modules are placed on 1 to *i*-

1, and n_1-k defective modules are placed on i+1 to N) is $\frac{C_{i-1}^{k-1} \times C_{N-i}^{n_1-k}}{C_{N-1}^{n_1-1}}$; (3) the contribution value of position i with defective modules is k/i. Therefore, the total AP contribution value of upstream defective module on current position i is calculated as:

$$AP(i) = \sum_{k=1}^{i} \left(\frac{C_{i-1}^{k-1} \times C_{N-i}^{n_1-k}}{C_{N-1}^{n_1-1}} \times \frac{C_{N-1}^{n_1-1}}{C_N^{n_1}} \times \frac{k}{i} \right) = \sum_{k=1}^{i} \left(\frac{C_{i-1}^{k-1} \times C_{N-i}^{n_1-k}}{C_N^{n_1}} \times \frac{k}{i} \right) \quad (k \le i \le N - n_1 + k)$$

Further, we can derive the formula of AP as follows:

$$AP = \frac{1}{N} \sum_{i=1}^{N} (AP(i)) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{i} \left(\frac{C_{i-1}^{k-1} \times C_{N-i}^{n_1-k}}{C_N^{n_1}} \times \frac{k}{i} \right)$$

The reasoning process of RR formula. RR can be expressed as:

$$RR = \frac{1}{1} \frac{C_{N-1}^{n_1-1}}{C_N^{n_1}} + \frac{1}{2} \frac{C_{N-2}^{n_1-1}}{C_N^{n_1}} + \cdots + \frac{1}{N - n_1 + 1} \frac{C_{N-(N-n_1+1)}^{n_1-1}}{C_N^{n_1}}$$

The first summation term $\frac{1}{1} \frac{C_{N-1}^{n_1-1}}{C_N^{n_1}}$ indicates that if the first defective module appears in the first position, then the RR value is 1, and the probability of occurrence of this event is $\frac{C_{N-1}^{n_1-1}}{C_N^{n_1}}$. Because the total possibility is $C_N^{n_1}$ (regarding the defective modules as repeated events), the possibility of the current situation is to select n_1-1 positions from the subsequent N-1 positions to place the defective modules, that is, the total possibility is $C_{N-1}^{n_1-1}$. By analogy, the first defective module can appear at most in the N-n_1+1 position. After sorting out the above formulas, the results are as follows:

$$RR = \sum_{i=1}^{N-n_1+1} \left(\frac{1}{i} \times \frac{C_{N-i}^{n_1-1}}{C_N^{n_1}} \right)$$

Appendix D. Supplementary results for RQ2 (influence on prediction performance)

In Section 7.2 (RQ2), two model comparison schemes, "NC vs. CC" and "NN vs. NC", are respectively used to investigate the influence of inconsistent labels on the prediction performance of a defect prediction model and the influence on the performance estimation of a defect prediction model. In this section, we report the results of more indicators including *MCC* and *ACC* for reference.

D.1. NC vs. CC

Table 4~9, respectively, report the distributions of |*diff*| and |*pgr*| between the two model (NC and CC) with respect to nine performance evaluation indicators (i.e., F_1, *AUC*, *ER*, *RI*, *MCC*, *AP*, *RR*, P_{opt}, and *ACC*) for three classifiers (random forest (RF) [23], naive Bayes (NB) [24], and logistic regression (LR) [25]) respectively. In particular, the values in shown in the yellow background in the table indicate the maximum mean value of |*diff*| or |*pgr*| in six multi-version-project defect data sets. The values in shown in the green background in the table indicate the confidence interval of the corrected confidence level contains 0 (i.e. the difference between the indicators of the two models (NC and CC) is not significant).

As can be seen from tables 4~9, only for the *RR* indicator, when the classifier is Naive Bayes or Logistic

Regression, neither |*diff*| nor |*pgr*| indicators are significantly different from zero (i.e., the confidence interval of the corrected confidence level contains 0) on the ECLIPSE-2007 data set. In other words, on the ECLIPSE-2007 data set, there is no significant difference in the *RR* indicator between model NC and model CC. This is understandable because there are three factors. First, *RR* is the reciprocal of the rank of the first real defective instance. The calculation of *RR* does not consider whether the prediction of the model for other instances is correct or not, nor whether the real defective instance with the same rank between the two models are the same instance, but only considers whether the highest rank of real defect instances is the same. Therefore, compared with other indicators, the *RR* indicator is more likely to be less influenced by inconsistent labels (noise). Second, the inconsistent label rate (noise rate) of the ECLIPSE-2007 data set is the smallest, so it is more difficult to influence the calculation of *RR*. Third, the number of data points on the ECLIPSE-2007 data set is the least (only 3). Once the difference of *RR* on two data points is 0, even if the difference of *RR* on the other data point is very large, the data points with difference of 0 will be more easily sampled when sampling is used to reflect the difference of population (1000 bootstrap sampling). This makes the difference of *RR* is not significant from the perspective of sampling group. The third point is particularly evident in Table 9.

Table 4. Random Forest: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	3.87	0.71	0.99	2.82	4.85	3	908.33	2.14	2.83
	StdDev	2.27	0.41	0.86	2.51	2.8	1.57	1465.08	0.95	2.73
	95% CI [LL, UL]	[1.52,5.37]	[0.28,0.99]	[0.36,1.53]	[0.99,4.39]	[2.3,7.85]	[1.25,4.02]	[50,1758.33]	[1.26,2.76]	[0,4.64]
Metrics-Repo-2010	Mean	7.36	6.29	31.36	37.42	37.12	9.67	33.09	10.14	25.95
	StdDev	7.99	8.77	130.77	130.84	133.39	14.55	66.82	10.38	26.89
	95% CI [LL, UL]	[5.75,10.28]	[4.34,9.12]	[12.49,107.41]	[18.89,112.93]	[18.48,128.82]	[6.8,14.92]	[19.81,55.67]	[8.18,13.88]	[19.29,34.22]
JIRA-HA-2019	Mean	7.84	1.45	2.96	13.79	8.83	10.86	31.23	3.73	21.32
	StdDev	8.93	1.42	3.16	13.68	10.81	11.99	73.15	3.33	38.75
	95% CI [LL, UL]	[5.2,11.19]	[1.1,2.02]	[2.15,4.22]	[10.32,18.95]	[6.08,12.6]	[7.93,15.75]	[15.61,66.44]	[2.9,4.93]	[14.17,41.23]
JIRA-RA-2019	Mean	7.21	1.28	3.45	12.8	8.43	9.21	62.25	3.11	9.26
	StdDev	5.56	0.96	3.8	10.42	7.97	9.62	109.27	3.42	8.54
	95% CI [LL, UL]	[5.85,9.25]	[1.02,1.6]	[2.49,4.85]	[10.17,16.18]	[6.3,11.53]	[7.07,13.93]	[39.22,109.01]	[2.32,4.5]	[6.93,12.16]
MA-SZZ-2020	Mean	3.91	0.32	1.15	7.35	3.75	2.8	8.82	2	7.76
	StdDev	6.08	0.41	1.21	10.38	4.97	4.54	41.54	2.67	10.75
	95% CI [LL, UL]	[3.21,4.89]	[0.27,0.37]	[1,1.34]	[6.14,8.94]	[3.16,4.62]	[2.24,3.57]	[5.04,18.79]	[1.67,2.42]	[6.49,9.64]
IND-JLMIV+R-2020	Mean	6.68	0.66	1.73	10.63	6.1	8.58	28.49	2.41	10.2
	StdDev	7.06	0.77	2.27	10.83	6.19	11.52	95.85	2.58	12.78
	95% CI [LL, UL]	[5.44,8.26]	[0.52,0.83]	[1.38,2.24]	[8.87,13.35]	[5.09,7.48]	[6.59,11.1]	[16.53,65.43]	[1.98,2.98]	[8.01,12.89]
Range of mean	[minMean, maxMean]	[3.87,7.84]	[0.32,6.29]	[0.99,31.36]	[2.82,37.42]	[3.75,37.12]	[2.8,10.86]	[8.82,908.33]	[2,10.14]	[2.83,25.95]

Table 5. Random Forest: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	10.21	2.06	0.99	2.82	4.85	5.05	204.72	13.86	5.83
	StdDev	6.41	1.21	0.86	2.51	2.8	2.5	195.61	5.26	5.74
	95% CI [LL, UL]	[2.98,14.28]	[0.78,2.86]	[0.36,1.53]	[0.99,4.39]	[2.3,6.7]	[2.21,6.62]	[73.83,323.31]	[7.95,17.23]	[0,9.66]
Metrics-Repo-2010	Mean	79.21	20.13	31.36	37.42	37.12	22.33	130.72	121.31	129.11
	StdDev	284	32.83	130.77	130.84	133.39	23.71	381.28	267.21	243.23
	95% CI [LL, UL]	[27.3,240.43]	[13.63,31.9]	[12.49,107.41]	[18.89,112.93]	[18.48,128.82]	[17.17,29.96]	[59.36,280.29]	[74.2,237.27]	[84.96,247.81]
JIRA-HA-2019	Mean	10.81	3.48	2.96	13.79	8.83	13.31	61.22	82.22	61.55
	StdDev	12.29	3.48	3.16	13.68	10.81	13.84	192.58	241.57	117.97
	95% CI [LL, UL]	[7.44,15.93]	[2.61,4.96]	[2.15,4.23]	[10.37,18.96]	[6.08,12.6]	[9.77,18.77]	[24.94,176.65]	[30.27,211.09]	[36.14,122.53]
JIRA-RA-2019	Mean	11.2	3.16	3.45	12.8	8.43	11.97	159.11	28.72	63.66
	StdDev	9.18	2.38	3.8	10.42	7.97	13.95	458.57	50.6	120.62
	95% CI	[9.17,14.87]	[2.53,3.98]	[2.49,4.85]	[10.17,16.18]	[6.31,11.53]	[9.2,19.48]	[78.35,423.08]	[18.99,53.4]	[38.46,126.13]

	[LL, UL]									
MA-SZZ-2020	Mean	4.99	0.67	1.15	7.35	3.75	3.24	18.95	6.3	28
	StdDev	7.26	0.91	1.21	10.38	4.97	4.86	85.06	11.48	92.52
	95% CI [LL, UL]	[4.16,6.15]	[0.56,0.81]	[1,1.34]	[6.13,8.94]	[3.16,4.62]	[2.63,4.06]	[10.73,38.64]	[4.98,8.34]	[19.18,51.1]
IND-JLMIV+R-2020	Mean	8	1.44	1.73	10.63	6.1	9.94	96.95	6.92	18.92
	StdDev	8.51	1.77	2.27	10.83	6.19	13.81	448.31	9.24	30.72
	95% CI [LL, UL]	[6.53,9.97]	[1.13,1.85]	[1.38,2.24]	[8.92,13.37]	[5.09,7.41]	[7.51,12.89]	[35.86,252.76]	[5.5,9.07]	[14.06,25.79]
Range of mean	[minMean, maxMean]	[4.99,79.21]	[1.44,20.13]	[0.99,31.36]	[2.82,37.42]	[3.75,37.12]	[3.24,22.33]	[18.95,204.72]	[6.3,121.31]	[5.83,129.11]

Table 6. Naive Bayes: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	4.57	0.75	3.28	8.03	6.24	2.78	22.22	0.74	0.92
	StdDev	0.71	0.35	1.71	3.24	1.7	2.03	38.49	0.91	1.59
	95% CI [LL, UL]	[3.81,5.04]	[0.36,0.97]	[2.14,4.32]	[5.55,10.08]	[4.7,7.36]	[0.5,4.08]	[0,44.44]	[0.11,1.3]	[0,1.84]
Metrics-Repo-2010	Mean	7.73	4.96	22.25	28.41	23.03	9.47	21.02	15.78	39.32
	StdDev	7.38	7.52	51.36	51.99	32.72	14.42	70.83	16.99	75.36
	95% CI [LL, UL]	[6.07,10.03]	[3.46,7.74]	[14.06,47.16]	[20.05,54]	[16.89,35.16]	[6.5,14.1]	[7.28,52.06]	[11.92,20.53]	[24.96,70.36]
JIRA-HA-2019	Mean	8.74	1.37	5.71	18.61	7.6	6.86	36.03	3.22	15.08
	StdDev	9.86	1.49	7.85	22.29	7.93	6.14	56.52	3.33	14.3
	95% CI [LL, UL]	[6.02,12.64]	[1.01,2.03]	[3.77,9.66]	[13.33,27.33]	[5.7,10.4]	[5.3,9.54]	[21.79,55.38]	[2.38,4.64]	[11.61,21]
JIRA-RA-2019	Mean	6.82	0.94	3.74	13.18	6.46	5.7	32.79	3.27	9.6
	StdDev	4.45	1.06	3.29	8.44	4.68	6.02	47.5	4.32	9.93
	95% CI [LL, UL]	[5.49,8.2]	[0.7,1.35]	[2.88,4.86]	[10.69,15.58]	[5.12,7.93]	[4.22,7.83]	[21.03,49.32]	[2.31,5.16]	[7.3,13.63]
MA-SZZ-2020	Mean	4.69	0.82	4.28	11.37	6.42	6.58	11.85	4.09	12.02
	StdDev	4.33	0.75	4.76	12.23	5.63	10.09	35.63	4.05	11.28
	95% CI [LL, UL]	[4.14,5.33]	[0.72,0.94]	[3.73,5.12]	[9.89,13.35]	[5.66,7.27]	[5.55,8.32]	[7.8,17.98]	[3.59,4.67]	[10.48,13.55]
IND-JLMIV+R-2020	Mean	6.03	0.7	3.75	11.6	5.92	7.63	25.84	3.06	17.08
	StdDev	6.49	1.01	4.54	11.63	6.2	10.35	57.45	3.27	35.41
	95% CI [LL, UL]	[4.93,7.37]	[0.55,0.96]	[3.1,4.82]	[9.45,14.14]	[5.01,7.5]	[5.82,9.97]	[16.62,41.58]	[2.55,3.78]	[12.04,26.94]
Range of mean	[minMean, maxMean]	[4.57,8.74]	[0.7,4.96]	[3.28,22.25]	[8.03,28.41]	[5.92,23.03]	[2.78,9.47]	[11.85,36.03]	[0.74,15.78]	[0.92,39.32]

Table 7. Naive Bayes: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	9.98	2.24	3.28	8.03	6.24	4.49	30.08	2.31	1.43
	StdDev	1.57	1.14	1.71	3.24	1.7	3.28	52.1	2.63	2.47
	95% CI [LL, UL]	[8.35,11.02]	[1.01,2.99]	[2.14,4.32]	[5.55,10.08]	[4.7,7.36]	[0.81,6.58]	[0,60.16]	[0.44,3.94]	[0,2.85]
Metrics-Repo-2010	Mean	38.06	16.99	22.25	28.41	23.03	21.78	55.64	91.89	109.21
	StdDev	62.66	29.92	51.36	51.99	32.72	28.48	211.55	148.19	185.4
	95% CI [LL, UL]	[26.52,61.46]	[11.24,28.32]	[13.95,47.01]	[20.05,54]	[16.81,35.16]	[15.57,30.16]	[14.59,157.99]	[62.96,150.17]	[70.57,174.62]
JIRA-HA-2019	Mean	13.29	3.41	5.71	18.61	7.6	8.6	71.63	45.65	50.67
	StdDev	14.95	3.71	7.85	22.29	7.93	7.45	169.55	83.51	65.92
	95% CI [LL, UL]	[9.25,19.29]	[2.5,5.01]	[3.77,9.66]	[13.32,27.33]	[5.7,10.4]	[6.63,11.5]	[38.55,173.48]	[28.08,87.51]	[34.9,77.03]
JIRA-RA-2019	Mean	10.1	2.31	3.74	13.18	6.46	7.09	72.13	28.29	40.38
	StdDev	6.03	2.76	3.29	8.44	4.68	7.68	155.96	46.95	83.7
	95% CI [LL, UL]	[8.26,11.89]	[1.72,3.54]	[2.89,4.87]	[10.69,15.58]	[5.12,7.94]	[5.15,9.79]	[39.23,149.47]	[17.73,49.13]	[23.74,89.2]
MA-SZZ-2020	Mean	8.6	2.05	4.28	11.37	6.42	8.59	25.38	87.47	117.95
	StdDev	8.21	1.98	4.76	12.23	5.63	12.74	121.27	433.31	292.93
	95% CI [LL, UL]	[7.61,9.91]	[1.8,2.36]	[3.73,5.11]	[9.89,13.35]	[5.66,7.27]	[7.32,11.03]	[14.45,54.4]	[46.37,177.67]	[81.63,170.59]
IND-JLMIV+R-2020	Mean	8.5	1.73	3.75	11.6	5.92	9.51	77.18	32.22	111.99
	StdDev	9	2.78	4.54	11.63	6.2	13.71	331.87	104.71	204.07
	95% CI [LL, UL]	[6.92,10.42]	[1.31,2.44]	[3.08,4.77]	[9.52,14.21]	[5.01,7.5]	[7.25,12.55]	[37.06,202.23]	[20.53,73.96]	[81.18,177.16]
Range of mean	[minMean, maxMean]	[8.5,38.06]	[1.73,16.99]	[3.28,22.25]	[8.03,28.41]	[5.92,23.03]	[4.49,21.78]	[25.38,77.18]	[2.31,91.89]	[1.43,119.99]

Table 8. Logistic Regression: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	2.96	2.09	3.11	6.48	4.64	1.48	50	3.29	6.5
	StdDev	3.35	1.54	4.66	9.37	5.93	1.9	50	4.29	4.61
	95% CI [LL, UL]	[0.08,5.14]	[0.43,3.11]	[0.36,5.82]	[0.9,17.31]	[0.2,11.37]	[0.37,3.67]	[0,83.33]	[0.35,5.91]	[3.53,9.27]
Metrics-Repo-2010	Mean	10.26	5.55	23.07	31.06	26.78	6.83	16.6	29.8	46.43
	StdDev	10.12	7.11	33.12	43.24	34.8	8.7	31.07	70.46	128.79
	95% CI [LL, UL]	[8.02,13.41]	[4.09,7.79]	[16.27,35.52]	[22.99,48.7]	[19.92,39.01]	[5.02,9.91]	[9.49,25.34]	[17.83,66.67]	[25.33,104.18]
JIRA-HA-2019	Mean	8.53	2.13	11.37	22.66	13.71	12.02	44.44	4.47	23.61
	StdDev	11.13	3.61	35.93	49.92	30.94	23.11	85.36	6.36	40.02
	95% CI [LL, UL]	[6.15,14.17]	[1.38,4]	[4.95,36]	[13.11,57.47]	[8.19,37.47]	[6.74,23.85]	[23.93,80.12]	[3.1,7.97]	[14.13,40.42]
JIRA-RA-2019	Mean	9.81	2.86	7.48	40.67	13.48	9.76	5.95	4.57	12.93
	StdDev	27.96	5.46	23.59	190.07	40.82	10.6	19.76	6.33	19.98
	95% CI [LL, UL]	[5.03,27.02]	[1.81,5.73]	[3.46,21.58]	[10.56,186.93]	[6.58,38.17]	[7.03,13.79]	[1.19,13.1]	[3.13,6.96]	[8.75,21.58]
MA-SZZ-2020	Mean	5.04	1.23	3.35	9.07	7.22	6.38	20.89	5.57	33.04
	StdDev	5.02	1.34	3.05	8.34	6.05	8.73	43.5	7.82	96.13
	95% CI [LL, UL]	[4.41,5.79]	[1.08,1.46]	[2.98,3.8]	[8.08,10.38]	[6.4,8.03]	[5.44,8.04]	[15.44,28.23]	[4.65,6.89]	[23.84,53.03]
IND-JLMIV+R-2020	Mean	5.95	0.94	2.72	9.44	7.54	8.23	20.47	6.27	20.88
	StdDev	6.76	1.38	3	10.51	8.32	11.26	47.62	5.28	28.48
	95% CI [LL, UL]	[4.76,7.44]	[0.75,1.34]	[2.23,3.39]	[7.68,11.87]	[6.2,9.61]	[6.58,11.04]	[14.06,34.35]	[5.38,7.43]	[16.42,27.12]
Range of mean	[minMean, maxMean]	[2.96,10.26]	[0.94,5.55]	[2.72,23.07]	[6.48,40.67]	[4.64,26.78]	[1.48,12.02]	[5.95,20.89]	[3.29,29.8]	[6.5,46.43]

Table 9. Logistic Regression: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	6.65	6.74	3.11	6.48	4.64	2.56	94.4	16.09	12.1
	StdDev	7.19	5.07	4.66	9.37	5.93	3.33	106.19	20.49	12.23
	95% CI [LL, UL]	[0.19,11.38]	[1.3,10.08]	[0.36,5.82]	[0.9,11.95]	[0.2,8.36]	[0.63,4.49]	[0,164.2]	[1.73,39.55]	[4.69,26.21]
Metrics-Repo-2010	Mean	58.23	21.08	23.07	31.06	26.78	19.61	119.23	231.67	85.32
	StdDev	103.31	30.56	33.12	43.24	34.8	26.06	357.16	1144.05	332.17
	95% CI [LL, UL]	[38.97,98.6]	[14.09,31.42]	[15.93,33.59]	[23.53,48.29]	[19.65,37.42]	[14.32,28.24]	[48.17,242.83]	[68.48,1007.07]	[36.58,255.87]
JIRA-HA-2019	Mean	20.59	10.88	11.37	22.66	13.71	25.34	78.65	117.3	75.93
	StdDev	54.63	38.59	35.93	49.92	30.94	85.75	183.28	357.94	153.62
	95% CI [LL, UL]	[10.87,65.81]	[4.22,36.14]	[5.11,33.93]	[13.68,61.88]	[8.21,37.3]	[9.89,93.45]	[41.11,171.79]	[43,343.68]	[42.6,151.26]
JIRA-RA-2019	Mean	21.99	8.25	7.48	40.67	13.48	12.84	6.81	33.65	116.2
	StdDev	84.14	18.19	23.59	190.07	40.82	15.31	22.66	50.12	252.75
	95% CI [LL, UL]	[8.33,85.77]	[4.34,16.95]	[3.58,22.7]	[10.74,158.92]	[6.57,37.79]	[8.97,18.37]	[2.62,17.96]	[21.93,51.87]	[56.81,227.89]
MA-SZZ-2020	Mean	8.94	3.25	3.35	9.07	7.22	9.16	89.12	78.04	162.87
	StdDev	8	3.91	3.05	8.34	6.05	12.35	360.47	361.51	598.63
	95% CI [LL, UL]	[7.96,10.01]	[2.82,3.94]	[2.95,3.78]	[8.08,10.31]	[6.42,8.12]	[7.89,11.55]	[54.21,162.15]	[47.44,181.63]	[105.1,305.89]
IND-JLMIV+R-2020	Mean	8.31	2.35	2.72	9.44	7.54	10.53	44.31	62.4	115.69
	StdDev	9.12	3.83	3	10.51	8.32	16	176.2	98.03	220.44
	95% CI [LL, UL]	[6.73,10.31]	[1.85,3.58]	[2.23,3.39]	[7.68,11.87]	[6.19,9.6]	[8.21,14.92]	[23.86,116.21]	[47.91,88.25]	[81.46,174.28]
Range of mean	[minMean, maxMean]	[6.65,58.23]	[2.35,21.08]	[2.72,23.07]	[6.48,40.67]	[4.64,26.78]	[2.56,25.34]	[6.81,119.23]	[16.09,231.67]	[12.1,162.87]

D.2. NN vs. NC

Tables 10~15, report the distributions of |*diff*| and |*pgr*| between the two model (NN and NC) with respect to nine performance evaluation indicators (i.e., F_1, *AUC*, *ER*, *RI*, *MCC*, *AP*, *RR*, P_{opt}, and *ACC*) for three classifiers (random forest (RF), naive Bayes (NB), and logistic regression (LR)) respectively.

From Tables 10~15, it is not difficult to find that the three observations to "NC vs. CC" (RQ2 in Section 7.2) are all applicable to "NN vs. NC". The only difference is that for the *RR* indicator, "NN vs. NC" shows no significant difference on 3 or 4 data sets (count the cells shown in green background). However, "NC vs. CC" only shows no

significant difference on the ECLIPSE-2007 data set (when the classifier is Naive Bayes or Logistic Regression). This shows that the *RR* indicator is more susceptible to inconsistent labels in test set than inconsistent labels in training set. The reason why the *RR* indicator between NN and NC models does not show significant differences on some data sets is the same as the three reasons analyzed in Appendix D.1.

(1) For "NC vs. CC", inconsistent labels may lead to model performance improvement or degradation (compared with clean data). Only if inconsistent labels consistently benefit to improve model performance on all version pairs, inconsistent labels should not be removed or handled in practice. (2) For "NN vs. NC", inconsistent labels will lead to overestimation or underestimation, and their existence will only lead to biased model evaluation. Combining (1) and (2), it is reasonable to remove the inconsistent labels, especially considering that inconsistent labels may mislead the interpretation of a defect prediction model.

Table 10. Random Forest: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	2.63	2.37	2.51	6.79	4.67	1.47	0	4.36	5.38
	StdDev	0.43	0.45	0.14	0.27	0.34	0.52	0	0.44	1.05
	95% CI [LL, UL]	[2.13,2.89]	[1.85,2.64]	[2.4,2.6]	[6.62,7.1]	[4.41,5.06]	[0.9,1.81]	[0,0]	[3.86,4.64]	[4.54,6.05]
Metrics-Repo-2010	Mean	15.41	10.22	47.76	55.53	36.33	12.36	10.03	13.63	16.59
	StdDev	13.92	9.94	83.36	82.26	52.12	14.53	23.93	14.33	17.91
	95% CI [LL, UL]	[12.25,19.56]	[7.92,13.31]	[31.18,78.21]	[38.34,81.44]	[26.14,56.05]	[9.05,16.82]	[4.71,17.88]	[10.69,18.09]	[12.93,23.33]
JIRA-HA-2019	Mean	6.93	5.01	3.68	15.12	8.89	9.29	7.69	8.05	12.48
	StdDev	7.73	4.19	3.5	13.32	7.58	9.52	48.04	7.11	11.58
	95% CI [LL, UL]	[5.14,10.1]	[3.75,6.48]	[2.83,5.08]	[11.43,19.92]	[6.8,11.73]	[6.82,12.72]	[0,23.08]	[6.22,10.63]	[9.45,16.71]
JIRA-RA-2019	Mean	6.03	4.18	4.32	16.06	9.92	8.67	0	7.56	14.25
	StdDev	5.79	3.64	3.9	13.81	8.09	6.82	0	6.62	12.04
	95% CI [LL, UL]	[4.51,7.95]	[3.18,5.34]	[3.27,5.63]	[12.14,20.56]	[7.43,12.46]	[6.65,10.95]	[0,0]	[5.82,9.64]	[10.71,18.39]
MA-SZZ-2020	Mean	3.01	1.03	2.07	7.76	3.69	4.12	2.3	1.81	4.4
	StdDev	4.19	1.46	2.31	7.91	4.95	7.83	12.04	2.39	6.23
	95% CI [LL, UL]	[2.47,3.78]	[0.83,1.27]	[1.71,2.41]	[6.57,9.02]	[2.98,4.61]	[3.17,5.61]	[0.63,4.59]	[1.44,2.16]	[3.45,5.36]
IND-JLMIV+R-2020	Mean	5.34	2.61	2.09	11.63	5.83	5.99	4.48	5.49	10.71
	StdDev	9.32	2.9	2.21	8.83	5.88	4.85	30.97	5.12	8.88
	95% CI [LL, UL]	[3.89,7.66]	[2.12,3.27]	[1.69,2.53]	[10.02,13.51]	[4.83,7.18]	[5.07,6.97]	[0.95,16.43]	[4.54,6.49]	[9.21,12.39]
Range of mean	[minMean, maxMean]	[2.63,15.41]	[1.03,10.22]	[2.07,47.76]	[6.79,55.53]	[3.69,36.33]	[1.47,12.36]	[0,10.03]	[1.81,13.63]	[4.4,16.59]

Table 11. Random Forest: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	13.4	6.89	2.51	6.79	4.67	5.88	6.33	26.44	10.74
	StdDev	4.74	1.07	0.14	0.27	0.34	0.48	5.8	5.74	3.69
	95% CI [LL, UL]	[8.33,16.53]	[5.67,7.54]	[2.4,2.6]	[6.62,7.05]	[4.41,4.89]	[5.33,6.18]	[1.19,10.14]	[21.22,30.23]	[8.25,14.98]
Metrics-Repo-2010	Mean	86.92	37.73	47.76	55.53	36.33	48.36	56.3	172.19	87.67
	StdDev	148.64	59.86	83.36	82.26	52.12	53.83	127.8	507.71	169.08
	95% CI [LL, UL]	[60.48,146.45]	[27.23,61.7]	[29.83,70.79]	[39.41,84.89]	[26.29,56.17]	[36.63,65.07]	[32.56,105.53]	[94.04,549.32]	[55.59,149.47]
JIRA-HA-2019	Mean	12.24	11.68	3.68	15.12	8.89	12.94	27.5	61.81	32.13
	StdDev	10.44	9.35	3.5	13.32	7.58	11.71	107.7	90.6	38.39
	95% CI [LL, UL]	[9.26,15.73]	[9.11,14.99]	[2.82,4.99]	[11.71,20.29]	[6.91,11.59]	[9.89,17.57]	[3.56,91.83]	[40.08,97.1]	[23.86,49.66]
JIRA-RA-2019	Mean	12.73	9.87	4.32	16.06	9.92	13.73	5.59	126.65	69.15
	StdDev	12.8	8.6	3.9	13.81	8.09	10.28	11.34	391.16	102.97
	95% CI [LL, UL]	[9.41,17.42]	[7.52,12.77]	[3.22,5.53]	[11.52,20.3]	[7.82,12.59]	[10.54,16.96]	[3.34,12.43]	[40.02,354.64]	[45.38,115.66]
MA-SZZ-2020	Mean	4.72	2.11	2.07	7.76	3.69	5.76	6.42	6.09	9.29
	StdDev	5.88	2.96	2.31	7.91	4.95	9.63	22.53	10.12	14.98
	95% CI [LL, UL]	[3.92,5.75]	[1.71,2.71]	[1.74,2.46]	[6.5,8.98]	[2.97,4.6]	[4.54,7.61]	[3.72,10.91]	[4.74,7.89]	[7.37,12.22]
IND-JLMIV+R-2020	Mean	9.38	5.69	2.09	11.63	5.83	9.26	14.03	14.75	19.28
	StdDev	7.56	6.63	2.21	8.83	5.88	6.98	38.13	15.15	23.73
	95% CI	[8.12,11.06]	[4.62,7.29]	[1.75,2.61]	[10.05,13.61]	[4.85,7.1]	[8.01,10.62]	[9.15,31.02]	[12.25,17.91]	[15.9,25.29]

	[LL, UL]									
Range of mean	[minMean, maxMean]	[4.72,86.92]	[2.11,37.73]	[2.07,47.76]	[6.79,55.53]	[3.69,36.33]	[5.76,48.36]	[5.59,56.3]	[6.09,172.19]	[9.29,87.67]

Table 12. Naive Bayes: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	1.59	1.87	2.56	6.18	4.14	1.16	0	3.91	5
	StdDev	0.73	0.52	0.42	0.3	0.3	0.48	0	0.51	0.5
	95% CI [LL, UL]	[0.76,2.03]	[1.27,2.18]	[2.19,2.83]	[5.88,6.38]	[3.84,4.34]	[0.63,1.47]	[0,0]	[3.34,4.24]	[4.48,5.33]
Metrics-Repo-2010	Mean	11.74	7.66	86.11	95.84	51.24	7.59	0	13.68	14.76
	StdDev	12.25	7.29	334.27	336.93	151.68	10.51	0	16.78	14.34
	95% CI [LL, UL]	[9.11,15.52]	[6.12,9.97]	[38.92,302.82]	[45.28,279.52]	[28.52,148.39]	[5.52,11.57]	[0,0]	[10.1,18.79]	[11.79,19.34]
JIRA-HA-2019	Mean	4.74	5.16	4.82	16.49	10.1	9.46	0	8.78	16.25
	StdDev	6.11	4.03	3.69	13.81	7.4	8.21	0	7.6	16.22
	95% CI [LL, UL]	[3.39,7.21]	[4.03,6.6]	[3.86,6.16]	[13.1,21]	[7.89,12.45]	[7.24,12.6]	[0,0]	[6.65,11.66]	[12.03,22.29]
JIRA-RA-2019	Mean	4.71	3.8	3.6	13.53	7.73	6.96	0.83	7.47	11.59
	StdDev	5.98	3.39	3.36	13	7.66	5.94	5.27	6.62	10.72
	95% CI [LL, UL]	[3.12,7.13]	[2.86,4.96]	[2.74,4.71]	[9.91,18.11]	[5.63,10.48]	[5.32,9.14]	[0,2.5]	[5.62,9.89]	[8.66,15.14]
MA-SZZ-2020	Mean	2.47	1.32	2.63	5.54	4	2.9	1.78	2.56	4.29
	StdDev	2.35	1.64	3.39	6.52	4.43	2.93	10.7	3.18	5.58
	95% CI [LL, UL]	[2.13,2.85]	[1.07,1.58]	[2.19,3.21]	[4.55,6.61]	[3.34,4.7]	[2.48,3.4]	[0.63,4.4]	[2.08,3.1]	[3.51,5.23]
IND-JLMIV+R-2020	Mean	6.91	2.51	3.79	10.71	5.9	5.22	3.1	5.55	11.23
	StdDev	12.33	2.37	3.79	8.32	6.92	4.33	15.03	4.2	9.22
	95% CI [LL, UL]	[5.01,10.15]	[2.13,3.02]	[3.26,4.62]	[9.28,12.34]	[4.72,7.55]	[4.4,6.1]	[1.03,8.06]	[4.82,6.4]	[9.43,13.06]
Range of mean	[minMean, maxMean]	[1.59,11.74]	[1.32,7.66]	[2.56,86.11]	[5.54,95.84]	[4,51.24]	[1.16,9.46]	[0,3.1]	[2.56,13.68]	[4.29,16.25]

Table 13. Naive Bayes: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	8.48	5.5	2.56	6.18	4.14	4.83	4.97	13.71	7
	StdDev	1.25	1.18	0.42	0.3	0.3	0.4	6.26	3.91	1.23
	95% CI [LL, UL]	[7.1,9.29]	[4.14,6.18]	[2.19,2.83]	[5.88,6.38]	[3.84,4.34]	[4.39,5.1]	[1.36,8.59]	[9.27,16.16]	[6.05,8.38]
Metrics-Repo-2010	Mean	92.73	26.18	86.11	95.84	51.24	31.66	19.05	371.36	80.29
	StdDev	194	29.01	334.27	336.93	151.68	33.66	25.74	2059.54	191.79
	95% CI [LL, UL]	[58.77,168.44]	[20.23,34.88]	[38.54,264.87]	[46.68,299.75]	[27.73,136.51]	[24.74,42.51]	[13.32,26.74]	[83.38,2049.23]	[47.23,165.24]
JIRA-HA-2019	Mean	11.65	12.44	4.82	16.49	10.1	14.33	3.41	84.42	52.87
	StdDev	8.95	9.35	3.69	13.81	7.4	10.7	3.43	135.46	82.82
	95% CI [LL, UL]	[9.11,15]	[9.71,15.83]	[3.85,6.17]	[12.5,20.9]	[8.11,12.63]	[11.33,18.04]	[2.61,4.66]	[52.88,152.57]	[35.52,103.74]
JIRA-RA-2019	Mean	10.21	8.89	3.6	13.53	7.73	11.25	5.68	52.79	43.79
	StdDev	12.25	7.88	3.36	13	7.66	9.04	11.12	75.04	87.53
	95% CI [LL, UL]	[6.89,14.29]	[6.58,11.4]	[2.7,4.74]	[9.69,17.66]	[5.62,10.53]	[8.46,14.14]	[3.47,11.94]	[35.36,89.01]	[28.06,100.88]
MA-SZZ-2020	Mean	5.53	3.28	2.63	5.54	4	5.26	7.27	230.44	33.33
	StdDev	6.21	4.11	3.39	6.52	4.43	5.4	41.84	2652.38	113.55
	95% CI [LL, UL]	[4.67,6.66]	[2.7,3.93]	[2.16,3.24]	[4.68,6.62]	[3.37,4.75]	[4.43,6.1]	[3.34,23.7]	[17.61,1280.28]	[22.57,74.03]
IND-JLMIV+R-2020	Mean	11.05	5.99	3.79	10.71	5.9	9.98	13.72	31.7	50.42
	StdDev	8.42	6.13	3.79	8.32	6.92	6.18	31.51	41.71	43.2
	95% CI [LL, UL]	[9.78,13.44]	[4.97,7.32]	[3.17,4.66]	[9.27,12.53]	[4.79,7.57]	[8.89,11.2]	[9.16,22.8]	[26.18,44.57]	[43.23,60.87]
Range of mean	[minMean, maxMean]	[5.53,92.73]	[3.28,26.18]	[2.56,86.11]	[5.54,95.84]	[4,51.24]	[4.83,31.66]	[3.41,19.05]	[13.71,371.36]	[7,80.29]

Table 14. Logistic Regression: distribution of the prediction performance |*diff*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	1.28	1.41	2.23	5.14	2.98	0.46	0	2.78	4.86
	StdDev	0.29	0.2	0.13	0.83	0.74	0.04	0	0.09	0.63
	95% CI [LL, UL]	[0.98,1.47]	[1.19,1.54]	[2.1,2.31]	[4.2,5.65]	[2.15,3.45]	[0.41,0.49]	[0,0]	[2.7,2.84]	[4.5,5.23]
Metrics-Repo-2010	Mean	11.08	6.64	109.88	119.58	59.04	7.71	12.87	13.27	14.04
	StdDev	11.89	6.91	475.06	479.46	185.97	11.81	57.22	17.1	15.16
	95% CI	[8.53,15.27]	[5.17,8.9]	[40.71,425.3]	[48.68,403.52]	[31.21,157.17]	[5.57,12.74]	[3.51,44.34]	[9.56,18.64]	[10.66,18.6]

	[LL, UL]									
JIRA-HA-2019	Mean	5.72	4.9	5.42	14.9	9.76	8.03	0	7.97	14.41
	StdDev	8.98	4.16	4.13	12.5	6.41	7.32	0	7.34	13.74
	95% CI [LL, UL]	[3.76,10.25]	[3.78,6.39]	[4.43,6.92]	[11.64,19.36]	[7.89,12]	[6.17,10.74]	[0,0]	[6.01,10.62]	[10.73,19.13]
JIRA-RA-2019	Mean	5	4.44	4.26	13.12	7.48	6.91	0	7.1	13.19
	StdDev	5.05	5.18	5.32	13.39	8.07	6.36	0	6.8	15.93
	95% CI [LL, UL]	[3.74,6.94]	[3.2,6.54]	[3.07,6.25]	[9.7,17.66]	[5.13,10.01]	[5.16,8.9]	[0,0]	[5.17,9.33]	[9.61,21.15]
MA-SZZ-2020	Mean	2.18	1.36	2.71	5.77	3.82	2.79	0.57	2.6	5.85
	StdDev	2.17	1.5	3.18	6.12	4.16	2.58	3.87	3.05	6.85
	95% CI [LL, UL]	[1.86,2.56]	[1.14,1.63]	[2.29,3.3]	[4.85,6.84]	[3.23,4.54]	[2.41,3.27]	[0.16,1.5]	[2.21,3.11]	[4.86,6.96]
IND-JLMIV+R-2020	Mean	5.98	2.25	3.85	11.38	5.72	7.62	10.72	5.46	13.99
	StdDev	11.04	2.29	3.37	8.89	6.37	16.01	48.55	4.17	13.57
	95% CI [LL, UL]	[4.37,8.96]	[1.85,2.77]	[3.24,4.55]	[9.71,13.24]	[4.62,7.04]	[5.38,11.29]	[4.35,24.86]	[4.73,6.3]	[11.66,17.08]
Range of mean	[minMean, maxMean]	[1.28,11.08]	[1.36,6.64]	[2.23,109.88]	[5.14,119.58]	[2.98,59.04]	[0.46,8.03]	[0,10.72]	[2.6,13.27]	[4.86,14.41]

Table 15. Logistic Regression: Distribution of the absolute performance gain ratio |*pgr*| in CVDP

Dataset		Classification scenario					Ranking scenario			
		F_1	*AUC*	*ER*	*RI*	*MCC*	*AP*	*RR*	P_{opt}	*ACC*
ECLIPSE-2007	Mean	6.82	4.66	2.23	5.14	2.98	4.12	2.58	12.2	7.19
	StdDev	3.12	0.38	0.13	0.83	0.74	0.18	2.26	3.08	1.28
	95% CI [LL, UL]	[3.34,8.83]	[4.23,4.9]	[2.1,2.31]	[4.2,5.65]	[2.15,3.45]	[3.92,4.23]	[1.19,3.91]	[8.7,13.75]	[6.41,7.95]
Metrics-Repo-2010	Mean	85.79	38.18	109.88	119.58	59.04	42.48	38.63	669.81	83.76
	StdDev	184.85	108.65	475.06	479.46	185.97	79.56	84.88	3676.07	187
	95% CI [LL, UL]	[53.91,163.56]	[21.8,107.53]	[42.05,378.26]	[49.42,369.72]	[30.81,141.88]	[29.39,88.54]	[23.04,74.48]	[140.09,3049.32]	[43.88,146.68]
JIRA-HA-2019	Mean	11	12.33	5.42	14.9	9.76	13.18	4.27	56.61	89.5
	StdDev	8.12	9.48	4.13	12.5	6.41	10.32	5.82	55.95	313.79
	95% CI [LL, UL]	[8.73,13.71]	[9.78,15.7]	[4.41,6.88]	[11.37,19.64]	[8.08,12.03]	[10.35,17.07]	[2.91,6.78]	[43.7,81.19]	[35.65,342.25]
JIRA-RA-2019	Mean	9.96	13.25	4.26	13.12	7.48	11.69	2.66	33.93	37.6
	StdDev	10.89	26.58	5.32	13.39	8.07	10.2	3.02	35.11	46.2
	95% CI [LL, UL]	[7.04,13.76]	[8.06,29.44]	[3,6.43]	[9.31,17.78]	[5.2,10.2]	[8.7,15.1]	[1.86,3.76]	[25.56,47.18]	[27.12,58.07]
MA-SZZ-2020	Mean	5.37	3.51	2.71	5.77	3.82	5.2	3.05	41.06	24.34
	StdDev	5.79	3.84	3.18	6.12	4.16	4.51	6.32	187.37	35.32
	95% CI [LL, UL]	[4.56,6.32]	[2.95,4.18]	[2.26,3.28]	[4.88,6.8]	[3.22,4.48]	[4.52,5.91]	[2.31,4.61]	[20.41,94.07]	[19.73,30.29]
IND-JLMIV+R-2020	Mean	11.51	5.46	3.85	11.38	5.72	13.07	23.5	45.5	66.61
	StdDev	7.89	6.09	3.37	8.89	6.37	15.38	56.13	56.48	88.01
	95% CI [LL, UL]	[10.24,13.33]	[4.47,6.82]	[3.24,4.52]	[9.86,13.05]	[4.67,7.04]	[10.96,17.52]	[15.87,42.09]	[37.82,62.53]	[53.18,87.52]
Range of mean	[minMean, maxMean]	[5.37,85.79]	[3.51,38.18]	[2.23,109.88]	[5.14,119.58]	[2.98,59.04]	[4.12,42.48]	[2.58,38.63]	[12.2,669.81]	[7.19,89.5]

Appendix E. Raw model performance values in RQ2

Figures 1 shows the overall distribution of the raw performance values (recall and FPR ($\frac{FP}{FP+TN}$)) of the baseline model (NN (built with Noise training data and applied to Noise test data), that is, the way to build the model in current literatures) on the LR (logistic regression) classifier. Figures 1 shows that except the ECLIPSE-2007 data set, the recalls of most data points (training set and test set pair) on all data sets are close to or more than 0.6, and the FPRs are lower than 0.3. The raw model performance values we obtained are in the performance range commonly found in current literatures (with practical value). These relatively high or good raw performance values demonstrate the basic predictive performance of the classifier, so comparisons between models ("NC vs. CC" and "NN vs. NC") can be considered reasonable. In other words, the experimental results and conclusions obtained in Section 7.2 (RQ2) are reasonable and of reference significance.

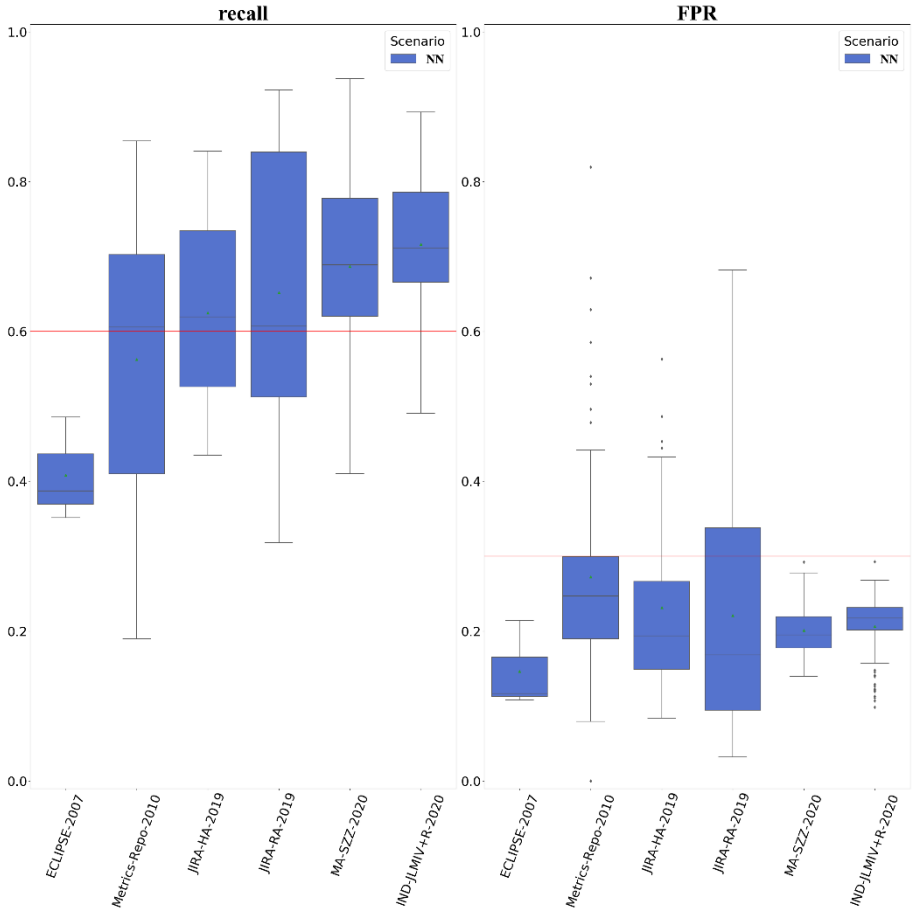

Fig. 1. The recall and FPR of NN, NC, and CC models (leaner: LR; feature selection: CFS; imbalance processing: SMOTE). Note: NB (naive Bayes) and RF (random forest) lead to a similar performance distribution as LR.

Appendix F. Can software metrics be used as a proxy of source code to identify inconsistent labels?

In TSILI, inconsistent labels are regarded as found if a cross-version instance has the same source code but different labels in different versions. During this process, there is a need to compare source code to identify cross-version instances. However, in practice, it is common to see that a data set only provides for each instance a number of software metrics (i.e. features) and a label indicating whether it is defective. In other words, source code is external information for a defect data set, which needs to be acquired additionally. In this context, an interesting question naturally arises: Can we use software metrics as a proxy of source code to identify cross-version instances? Indeed, in previous studies [26, 27, 28], it is not uncommon to see that software metric information is used to identify "inconsistent instances" in a defect data set. In [26, 28], if two instances in a version had identical values for all features but different labels, they were called "inconsistent instances". In [27], inconsistent cross-version instances were also examined. In their view, inconsistent instances are problematic in the context of machine learning and hence should be excluded when building and evaluating a defect prediction model. As can be seen, the concept of their inconsistent cross-version instances is very similar to the concept cross-version instances with "inconsistent labels" in our study. The difference is that the former uses software metrics rather than source code to identify cross-version instances. At a glance, it seems that we could use software metrics to replace source code to identify cross-version instances in our study. In fact, such a replacement is problematic due to the following two-fold reasons. On the one hand, the fact that two instances have identical source code does not necessarily mean that they have identical values for all features (i.e. metrics). The reason is that many metrics depend on the use context of an instance rather than only on its source code. For example, two functions with identical code may have different

values for the "called-by number" feature. On the other hand, the fact that two instances have identical values for all features does not necessarily mean that they have identical source code. For example, it is possible that two instances with different source code have identical values for all features. As a result, if we use software metrics as a proxy to identify cross-version instances, it is possible to miss real "inconsistent labels" or report false "inconsistent labels", thus leading to a low accuracy of inconsistent label identification. Given this situation, we should not use software metrics as a proxy to identify inconsistent labels.

We next use the inconsistent labels identified by TSILI as the ground truth to empirically understand how good software metrics can be used as a proxy for inconsistent label identification. For the simplicity of presentation, we use SMI to denote the method that uses software metrics to identify inconsistent labels. Fig. 2 reports for SMI the *precision*, *recall*, and F_1 on six multi-version-project data sets. Here, *precision* denotes the percentage of inconsistent labels identified by SMI that are also identified by TSILI, *recall* denotes the percentage of inconsistent labels identified by TSILI that are also identified by SMI, and F_1 is the harmonic mean of *precision* and *recall*. As can be seen, for five out of the six data sets, all the three indicators vary in a large range. For ECLIPSE-2007, all the three indicators have a small variance, as the inconsistent label ratios are low on all the three versions. Furthermore, on the one hand, for most data sets, the median/mean recall is low, indicating a high false negative (i.e., real inconsistent labels are incorrectly identified as non-inconsistent labels by SMI). On the other hand, for most data sets, the median/mean precision is around or less than 0.8, indicating there is non-negligible false positive (i.e., non-inconsistent labels are incorrectly identified as inconsistent labels by SMI). Overall, the above results reveal that SMI has a low application value, even though it has a lower computation cost than TSILI.

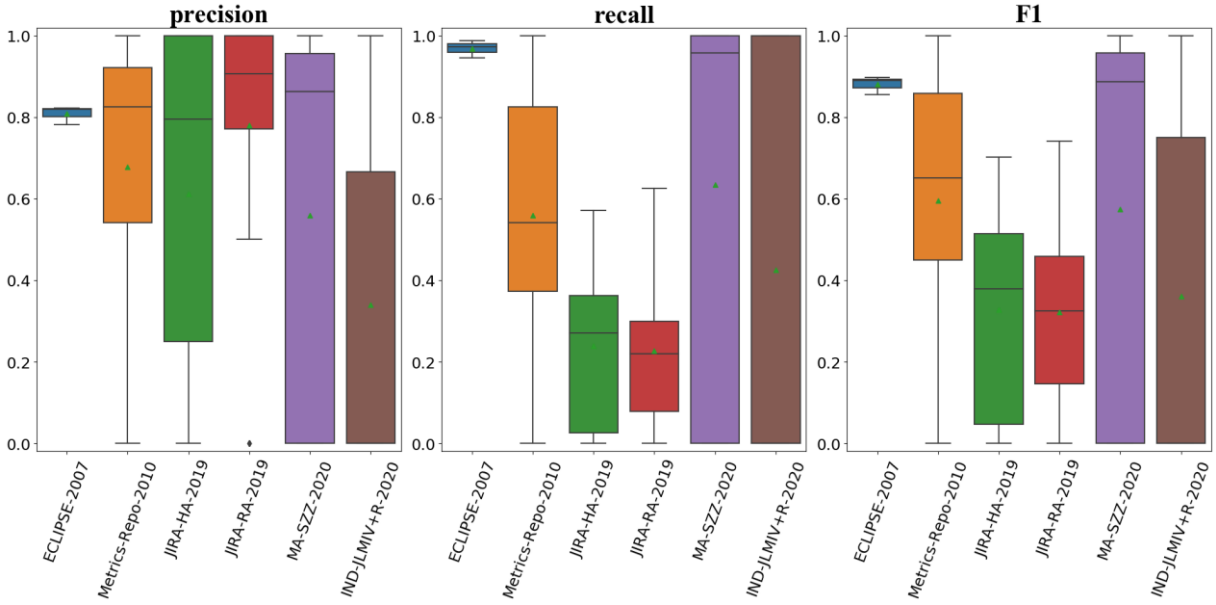

Fig. 2. Distribution of prediction performance scores of SMI for detecting inconsistent labels

References (A-F)

[1] M. Jureczko, D. Spinellis. Using object-oriented design metrics to predict software defects. RELCOMEX 2010: 69-81.

[2] S. Yatish, J. Jiarpakdee, P. Thongtanunam, C. Tantithamthavorn. Mining software defects: should we consider affected releases? ICSE 2019: 654-665.

[3] T. Zimmermann, R. Premraj, A. Zeller. Predicting defects for Eclipse. PROMISE 2007, article no 9: 1-7.

[4] S. Herbold, A. Trautsch, F. Trautsch. Issues with SZZ: an empirical assessment of the state of practice of defect prediction data collection. arXiv preprint arXiv:1911.08938v2, 2020.

[5] L.C. Briand, J. Wust, J.W. Daly, D.V. Porter. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software, 51(3), 2000: 245-273.

[6] Y. Zhou, B. Xu, H. Leung, L. Chen. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Transactions

on Software Engineering and Methodology, 23(1), 2014: 1-51.
[7] Y. Zhou, H.K.N Leung, B. Xu. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Transactions on Software Engineering, 35(5), 2009: 607-623.
[8] M.J. Shepperd, D. Bowes, T. Hall. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. IEEE Transactions on Software Engineering, 40(6), 2014: 603-616.
[9] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, F. Wu. Mutation-aware fault prediction. ISSTA 2016: 330-341.
[10] J. Yao, M.J. Shepperd. Assessing software defection prediction performance: why using the Matthews correlation coefficient matters. EASE 2020: 120-129.
[11] J. Han, M. Kamber, J. Pei. Data mining: concepts and techniques, 3rd edition. Morgan Kaufmann 2011, ISBN 978-0123814791.
[12] J. Huang, C.X. Ling. Using AUC and accuracy in evaluating learning algorithms. IEEE Transactions on knowledge and Data Engineering, 17(3), 2005: 299-310.
[13] Y. Tang, F. Zhao, Y. Yang, H. Lu, Y. Zhou, B. Xu. Predicting vulnerable components via text mining or software metrics? an effort-aware perspective. QRS 2015: 27-36.
[14] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 39(6), 2013: 757-773.
[15] T. Mende, R. Koschke. Revisiting the evaluation of defect prediction models. PROMISE 2009: 1–10.
[16] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. SIGSOFT FSE 2016: 157-168.
[17] C.D. Manning, P. Raghavan, H. Schütze. Introduction to information retrieval. Cambridge University Press 2008, ISBN 978-0-521-86571-5, pp. I-XXI, 1-482.
[18] Q. Huang, X. Xia, D. Lo, X. Chen, Q. Gu. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empirical Software Engineering, 24(5), 2019: 2823-2862.
[19] Q. Song, Z. Jia, M.J. Shepperd, S. Ying, J. Liu. A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering, 37(3), 2011: 356-370.
[20] S. Lai. Word and document embeddings based on neural network approaches. arXiv: 1611.05962, 2016. http://pdfs.semanticscholar.org/3e23/659a2de955727d548c9f26a044a59c3eb1b0.pdf (in Chinese).
[21] E. Arisholm, L. C. Briand, E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 83(1), 2010: 2–17.
[22] Y. Liu, Y. Li, J. Guo, Y. Zhou, B. Xu. Connecting software metrics across versions to predict defects. SANER 2018: 232-243.
[23] L. Breiman. Random forests. Machine learning, 45(1), 2001: 5–32.
[24] M. Kuhn. Building Predictive Models in R Using the caret Package. Journal of Statistical Software, 28(5), 2008: 1-26.
[25] S. Fritsch, F. Guenther. neuralnet: Training of neural networks. 2015. [Online]. Available: http://CRAN.R-project.org/package=neuralnet
[26] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. EASE 2011: 96-103.
[27] Z. Sun, J. Li, H. Sun. An empirical study of public data quality problems in cross project defect prediction. arXiv preprint arXiv:1805.10787, 2018.
[28] M. Shepperd, Q. Song, Z. Sun, C. Mair. Data quality: some comments on the NASA software defect datasets. IEEE Transactions on Software Engineering, 39(9), 2013: 1208-1215.

Appendix G. To what extent might previous studies be potentially influenced by inconsistent labels?

In this section, we answer this question by investigating the number of previous studies that used them as the subject data sets to conduct their experiments.

In order to avoid ambiguity, we separate the two concepts of "number of citations" and "number of literatures that had experimented with the collected data sets". The "number of citations" refers to the total number of citations by other literatures for the original papers that published the target multi-version-project defect data set. The "number of literatures that had experimented with the collected data sets" refers to the total number of other literatures, which not only cited the original papers of the target multi-version-project defect data set, but also used the target multi-version-project defect data set in their experiments. Generally, "number of citations" cannot be used as a proxy for the frequency of a data set usage, because other literatures may only introduce the methods or ideas of the cited paper. Therefore, we additionally count the "number of literatures that had experimented with the collected data sets" as a proxy for the frequency of a data set usage.

Table 16 (i.e., Table 9 in Section 8.4 in our manuscript) summarizes the "number of citations" and "number of literatures that had experimented with the collected data sets" for the existing multi-version-project defect data sets investigated in our study. The 1st column lists five existing multi-version-project defect data sets investigated in our study. The 2nd column lists the original paper(s) publishing each multi-version-project defect data set. The 3rd column reports how many other literatures cite the original literature, i.e., "number of citations" (reported by Google scholar, March 26, 2021). The 4th column reports the total number of other literatures (written in English) that use the corresponding data sets to conduct their experiments, i.e., "number of literatures that had experimented with the collected data sets" (inspected by the first author and confirmed by the seventh author). The 5th column reports the list number range of "literatures that had experimented with the collected data sets". Note that the Metrics-Repo-2010 data set was first published in [A2]. However, most literature cite [A3] and [A4] as its source. The reason was that [A3] was published in a well-known international conference on Predictive Models in Software Engineering (PROMISE), aiming to share publicly accessible data sets. In particular, Metrics-Repo-2010 was put on the corresponding promise repository website [A4]. Given this situation, we use them (i.e. [A2], [A3], and [A4]) as three sources to count the number of (different) citations. As can be seen, JIRA-RA-2019 (JIRA-HA-2019) and IND-JLMIV+R-2020 data sets were used by few studies (the "number of literatures that had experimented with the collected data sets" are 6 and 5 respectively), as they are two recently published multi-version-project defect data sets. However, ECLIPSE-2007 and Metrics-Repo-2010 data sets were widely used in previous studies (the "number of literatures that had experimented with the collected data sets" is 144 and 264 respectively). This indicates that inconsistent labels have a potentially wide influence on previous studies.

It is important to note that the influence of inconsistent labels on the existing literature may be overestimated, although the "number of literatures that had experimented with the collected data sets" is a more secure proxy for estimating the potential influence of inconsistent labels than the "number of citations". This is because our study does not conduct replication experiments to investigate the specific influence of inconsistent labels on each of the existing literatures (it is a large amount of work that could not be done in our study alone). It is still an open problem to investigate the actual influence of each multi-version-project defect data set with inconsistent labels on the existing literatures, pending an in-depth or extensive empirical study in the future.

Table 16. Literatures potentially influenced by target multi-version-project defect data sets

Defect data sets	Source	Number of citations	Number of literatures that had experimented with the collected data sets	List number range of literatures that had experimented with the collected data sets
ECLIPSE-2007	[A1]	817	144	[1~144]
Metrics-Repo-2010	[A2, A3, A4]	453	264	[145~408]
JIRA-HA-2019 / JIRA-RA-2019	[A5]	17	6	[409~414]
IND-JLMIV+R-2020	[A6]	9	5	[415~419]

The literatures of five target multi-version-project defect data sets

[A1] T. Zimmermann, R. Premraj, A. Zeller. Predicting defects for Eclipse. In Proceedings of the Third International Workshop on Predictor Models in Software Engineering, ser. PROMISE '07. IEEE Computer Society, 2007: 9–.

[A2] M. Jureczko, D. Spinellis. Using object-oriented design metrics to predict software defects. In Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010: 69-81.

[A3] M. Jureczko, L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, 2010: 1–10.

[A4] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan. The promise repository of empirical software engineering data, June 2012.

[A5] S. Yatish, J. Jiarpakdee, P. Thongtanunam, C. Tantithamthavorn. Mining software defects: should we consider affected releases? ICSE 2019: 654-665.

[A6] S. Herbold, A. Trautsch, F. Trautsch. Issues with SZZ: An empirical assessment of the state of practice of defect prediction data collection. arXiv preprint arXiv:1911.08938v2, 2020.

The list of citations using the ECLIPSE-2007 data set

[1] Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction[C]//Proceedings of the 30th international conference on Software engineering. ACM, 2008: 181-190.

[2] Zimmermann T, Nagappan N, Gall H, et al. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process[C]//Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009: 91-100.

[3] Bird C, Bachmann A, Aune E, et al. Fair and balanced?: bias in bug-fix datasets[C]//Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009: 121-130.

[4] Nagappan N, Zeller A, Zimmermann T, et al. Change bursts as defect predictors[C]//2010 IEEE 21st International Symposium on Software Reliability Engineering. IEEE, 2010: 309-318.

[5] Mende T, Koschke R. Effort-aware defect prediction models[C]//2010 14th European Conference on Software Maintenance and Reengineering. IEEE, 2010: 107-116.

[6] Tantithamthavorn C, McIntosh S, Hassan A E, et al. Automated parameter optimization of classification techniques for defect prediction models[C]//2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 2016: 321-332.

[7] Li M, Zhang H, Wu R, et al. Sample-based software defect prediction with active and semi-supervised learning[J]. Automated Software Engineering, 2012, 19(2): 201-230.

[8] Zhou Y, Xu B, Leung H. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems[J]. Journal of Systems and Software, 2010, 83(4): 660-674.

[9] Tantithamthavorn C, McIntosh S, Hassan A E, et al. An empirical comparison of model validation techniques for defect prediction models[J]. IEEE Transactions on Software Engineering, 2017, 43(1): 1-18.

[10] Zhang H. An investigation of the relationships between lines of code and defects[C]//2009 IEEE International Conference on Software Maintenance. IEEE, 2009: 274-283.

[11] Khoshgoftaar T M, Gao K, Seliya N. Attribute selection and imbalanced data: Problems in software defect prediction[C]//2010 22nd IEEE International Conference on Tools with Artificial Intelligence. IEEE, 2010, 1: 137-144.

[12] Shihab E, Jiang Z M, Ibrahim W M, et al. Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project[C]//Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2010: 4.

[13] Zhang H. On the distribution of software faults[J]. IEEE Transactions on Software Engineering, 2008, 34(2): 301-302.

[14] Bettenburg N, Hassan A E. Studying the impact of social structures on software quality[C]//2010 IEEE 18th International Conference on Program Comprehension. IEEE, 2010: 124-133.

[15] Wang H, Khoshgoftaar T M, Napolitano A. A comparative study of ensemble feature selection techniques for software defect prediction[C]//2010 Ninth International Conference on Machine Learning and Applications. IEEE, 2010: 135-140.

[16] Yang X, Tang K, Yao X. A learning-to-rank approach to software defect prediction[J]. IEEE Transactions on Reliability, 2015, 64(1): 234-246.

[17] Nguyen T H D, Adams B, Hassan A E. Studying the impact of dependency network measures on software quality[C]//2010 IEEE International Conference on Software Maintenance. IEEE, 2010: 1-10.

[18] Premraj R, Herzig K. Network versus code metrics to predict defects: A replication study[C]//2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, 2011: 215-224.

[19] Wang H, Khoshgoftaar T M, Seliya N. How many software metrics should be selected for defect prediction?[C]//Twenty-Fourth International FLAIRS Conference. 2011.

[20] Kpodjedo S, Ricca F, Galinier P, et al. Design evolution metrics for defect prediction in object oriented systems[J]. Empirical Software Engineering, 2011, 16(1): 141-175.

[21] Rodriguez D, Herraiz I, Harrison R. On software engineering repositories and their open problems[C]//2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE). IEEE, 2012: 52-56.

[22] Liu W, Liu S, Gu Q, et al. Empirical studies of a two-stage data preprocessing approach for software fault prediction[J]. IEEE Transactions on Reliability, 2016, 65(1): 38-53.

[23] Khoshgoftaar T M, Gao K, Napolitano A, et al. A comparative study of iterative and non-iterative feature selection techniques for software defect

prediction[J]. Information Systems Frontiers, 2014, 16(5): 801-822.
[24] Wang H, Khoshgoftaar T M, Van Hulse J. A comparative study of threshold-based feature selection techniques[C]//2010 IEEE International Conference on Granular Computing. IEEE, 2010: 499-504.
[25] Khoshgoftaar T M, Gao K, Napolitano A. An empirical study of feature ranking techniques for software quality prediction[J]. International journal of software engineering and knowledge engineering, 2012, 22(02): 161-183.
[26] Elish M O, Al-Yafei A H, Al-Mulhem M. Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse[J]. Advances in Engineering Software, 2011, 42(10): 852-859.
[27] Zhou Y, Xu B, Leung H, et al. An in-depth study of the potentially confounding effect of class size in fault prediction[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2014, 23(1): 10.
[28] Caglayan B, Bener A, Koch S. Merits of using repository metrics in defect prediction for open source projects[C]//2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. IEEE, 2009: 31-36.
[29] Wang H, Khoshgoftaar T M, Napolitano A. Software measurement data reduction using ensemble techniques[J]. Neurocomputing, 2012, 92: 124-132.
[30] Li W, Huang Z, Li Q. Three-way decisions based software defect prediction[J]. Knowledge-Based Systems, 2016, 91: 263-274.
[31] Shihab E. An exploration of challenges limiting pragmatic software defect prediction[D]. , 2012.
[32] Zeller A, Zimmermann T, Bird C. Failure is a four-letter word: a parody in empirical research[C]//Proceedings of the 7th International Conference on Predictive Models in Software Engineering. ACM, 2011: 5.
[33] Zimmerman T, Nagappan N, Herzig K, et al. An empirical study on the relation between dependency neighborhoods and failures[C]//2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. IEEE, 2011: 347-356.
[34] Krishnan S, Strasburg C, Lutz R R, et al. Are change metrics good predictors for an evolving software product line?[C]//Proceedings of the 7th International Conference on Predictive Models in Software Engineering. ACM, 2011: 7.
[35] Ibrahim W M, Bettenburg N, Adams B, et al. On the relationship between comment update practices and software bugs[J]. Journal of Systems and Software, 2012, 85(10): 2293-2304.
[36] Moser R, Pedrycz W, Succi G. Analysis of the reliability of a subset of change metrics for defect prediction[C]//Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. ACM, 2008: 309-311.
[37] Lu H, Kocaguneli E, Cukic B. Defect prediction between software versions with active learning and dimensionality reduction[C]//2014 IEEE 25th International Symposium on Software Reliability Engineering. IEEE, 2014: 312-322.
[38] Oyetoyan T D, Cruzes D S, Conradi R. A study of cyclic dependencies on defect profile of software components[J]. Journal of Systems and Software, 2013, 86(12): 3162-3182.
[39] Tantithamthavorn C, McIntosh S, Hassan A E, et al. The impact of automated parameter optimization on defect prediction models[J]. IEEE Transactions on Software Engineering, 2018:1-1.
[40] Rathore S S, Kumar S. Towards an ensemble based system for predicting the number of software faults[J]. Expert Systems with Applications, 2017, 82: 357-382.
[41] Wahyudin D, Ramler R, Biffl S. A framework for defect prediction in specific software project contexts[C]//IFIP Central and East European Conference on Software Engineering Techniques. Springer, Berlin, Heidelberg, 2008: 261-274.
[42] Gao K, Khoshgoftaar T M. Software Defect Prediction for High-Dimensional and Class-Imbalanced Data[C]//SEKE. 2011: 89-94.
[43] Wang H, Khoshgoftaar T M, Van Hulse J, et al. Metric selection for software defect prediction[J]. International Journal of Software Engineering and Knowledge Engineering, 2011, 21(02): 237-257.
[44] Pipitone J, Easterbrook S. Assessing climate model software quality: a defect density analysis of three models[J]. Geoscientific Model Development, 2012, 5(4): 1009-1022.
[45] Liu Y, Cheah W P, Kim B K, et al. Predict software failure-prone by learning Bayesian network[J]. International Journal of Advanced Science and Technology, 2008, 1(1): 35-42.
[46] Krishnan S, Strasburg C, Lutz R R, et al. Predicting failure-proneness in an evolving software product line[J]. Information and Software Technology, 2013, 55(8): 1479-1495.
[47] Tan X, Peng X, Pan S, et al. Assessing software quality by program clustering and defect prediction[C]//2011 18th working conference on Reverse Engineering. IEEE, 2011: 244-248.
[48] Chen J, Liu S, Liu W, et al. A two-stage data preprocessing approach for software fault prediction[C]//2014 Eighth International Conference on Software Security and Reliability (SERE). IEEE, 2014: 20-29.
[49] Marinescu C. Are the classes that use exceptions defect prone?[C]//Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution. ACM, 2011: 56-60.
[50] Gao K, Khoshgoftaar T M, Napolitano A. Impact of data sampling on stability of feature selection for software measurement data[C]//2011 IEEE 23rd International Conference on Tools with Artificial Intelligence. IEEE, 2011: 1004-1011.
[51] Rathore S S, Kumar S. Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems[J]. Knowledge-Based Systems, 2017, 119: 232-256.
[52] Marinescu R, Marinescu C. Are the clients of flawed classes (also) defect prone?[C]//2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation. IEEE, 2011: 65-74.
[53] Tantithamthavorn C, Hassan A E. An experience report on defect modelling in practice: Pitfalls and challenges[C]//Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. ACM, 2018: 286-295.
[54] Wang H, Khoshgoftaar T M, Napolitano A. An Empirical Study of Software Metrics Selection Using Support Vector Machine[C]//SEKE. 2011: 83-88.
[55] Gao K, Khoshgoftaar T M, Napolitano A. Combining Feature Subset Selection and Data Sampling for Coping with Highly Imbalanced Software Data[C]//SEKE. 2015: 439-444.
[56] Wang H, Khoshgoftaar T M, Wald R. Measuring robustness of feature selection techniques on software engineering datasets[C]//2011 IEEE International Conference on Information Reuse & Integration. IEEE, 2011: 309-314.
[57] Plosch R, Gruber H, Hentschel A, et al. On the relation between external software quality and static code analysis[C]//2008 32nd Annual IEEE Software Engineering Workshop. IEEE, 2008: 169-174.
[58] Marinescu C. Should we beware the exceptions? an empirical study on the eclipse project[C]//2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE, 2013: 250-257.
[59] Tantithamthavorn C, Hassan A E, Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models[J]. IEEE Transactions on Software Engineering, 2018.
[60] Liu W, Liu S, Gu Q, et al. Fecs: A cluster based feature selection method for software fault prediction with noises[C]//2015 IEEE 39th Annual Computer Software and Applications Conference. IEEE, 2015, 2: 276-281.
[61] Kuo C S, Huang C Y. A study of applying the bounded generalized pareto distribution to the analysis of software fault distribution[C]//2010 IEEE International Conference on Industrial Engineering and Engineering Management. IEEE, 2010: 611-615.
[62] Devine T, Goseva-Popstojanova K, Krishnan S, et al. Assessment and cross-product prediction of software product line quality: accounting for reuse across products, over multiple releases[J]. Automated Software Engineering, 2016, 23(2): 253-302.
[63] Zhang H, Cheung S C. A cost-effectiveness criterion for applying software defect prediction models[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 643-646.

[64] Zhang H, Wu R. Sampling program quality[C]//2010 IEEE International Conference on Software Maintenance. IEEE, 2010: 1-10.
[65] Wang H, Khoshgoftaar T M, Liang Q. A study of software metric selection techniques: stability analysis and defect prediction model performance[J]. International journal on artificial intelligence tools, 2013, 22(05): 1360010.
[66] Pipitone J. Software quality in climate modelling[J]. Master's thesis, Department of Computer Science, University of Toronto, 2010.
[67] Steff M, Russo B. Measuring architectural change for defect estimation and localization[C]//2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, 2011: 225-234.
[68] Choudhary G R, Kumar S, Kumar K, et al. Empirical analysis of change metrics for software fault prediction[J]. Computers & Electrical Engineering, 2018, 67: 15-24.
[69] Wang H, Khoshgoftaar T M, Napolitano A. An empirical investigation on wrapper-based feature selection for predicting software quality[J]. International Journal of Software Engineering and Knowledge Engineering, 2015, 25(01): 93-114.
[70] Wang H, Khoshgoftaar T M, Seliya N. On the stability of feature selection methods in software quality prediction: an empirical investigation[J]. International Journal of Software Engineering and Knowledge Engineering, 2015, 25(09n10): 1467-1490.
[71] Wang H, Khoshgoftaar T M, Napolitano A. Stability of filter-and wrapper-based software metric selection techniques[C]//Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014). IEEE, 2014: 309-314.
[72] Wang H, Khoshgoftaar T M, Napolitano A. An empirical study on the stability of feature selection for imbalanced software engineering data[C]//2012 11th International Conference on Machine Learning and Applications. IEEE, 2012, 1: 317-323.
[73] Khoshgoftaar T M, Gao K, Van Hulse J. Feature selection for highly imbalanced software measurement data[M]//Recent trends in information reuse and integration. Springer, Vienna, 2012: 167-189.
[74] GAO K, KHOSHGOFTAAR T M, WALD R. The use of under-and oversampling within ensemble feature selection and classification for software quality prediction[J]. International Journal of Reliability, Quality and Safety Engineering, 2014, 21(01): 1450004.
[75] Zhang H, Tan H B K, Marchesi M. The distribution of program sizes and its implications: An eclipse case study[C]//1st International Symposium on Emerging Trends in Software Metrics. 2009: 1-10.
[76] Khoshgoftaar T M, Gao K, Napolitano A. A Comparative Study of Different Strategies for Predicting Software Quality[C]//SEKE. 2011, 2011: 65-70.
[77] Mizuno O, Hata H. An empirical comparison of fault-prone module detection approaches: Complexity metrics and text feature metrics[C]//2010 IEEE 34th Annual Computer Software and Applications Conference. IEEE, 2010: 248-249.
[78] Wang H, Khoshgoftaar T M, Wald R, et al. A comparative study on the stability of software metric selection techniques[C]//2012 11th International Conference on Machine Learning and Applications. IEEE, 2012, 2: 301-307.
[79] Guo Y, Würsch M, Giger E, et al. An empirical validation of the benefits of adhering to the law of demeter[C]//2011 18th Working Conference on Reverse Engineering. IEEE, 2011: 239-243.
[80] Mizuno O, Hata H. An integrated approach to detect fault-prone modules using complexity and text feature metrics[M]//Advances in Computer Science and Information Technology. Springer, Berlin, Heidelberg, 2010: 457-468.
[81] Oyetoyan T D, Cruzes D S, Conradi R. Can refactoring cyclic dependent components reduce defect-proneness?[C]//2013 IEEE International Conference on Software Maintenance. IEEE, 2013: 420-423.
[82] Khoshgoftaar T M, Gao K, Napolitano A. An empirical study of predictive modeling techniques of software quality[C]//International Conference on Bio-Inspired Models of Network, Information, and Computing Systems. Springer, Berlin, Heidelberg, 2010: 288-302.
[83] Jiarpakdee J, Tantithamthavorn C, Treude C. AutoSpearman: Automatically Mitigating Correlated Software Metrics for Interpreting Defect Models[C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2018: 92-103.
[84] Wang H, Khoshgoftaar T M. Measuring stability of threshold-based feature selection techniques[C]//2011 IEEE 23rd International Conference on Tools with Artificial Intelligence. IEEE, 2011: 986-993.
[85] Gao K, Khoshgoftaar T M, Napolitano A. The use of ensemble-based data preprocessing techniques for software defect prediction[J]. International Journal of Software Engineering and Knowledge Engineering, 2014, 24(09): 1229-1253.
[86] Khoshgoftaar T M, Gao K, Van Hulse J. A novel feature selection technique for highly imbalanced data[C]//2010 IEEE International Conference on Information Reuse & Integration. IEEE, 2010: 80-85.
[87] Khoshgoftaar T M, Gao K, Napolitano A. Improving software quality estimation by combining feature selection strategies with sampled ensemble learning[C]//Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014). IEEE, 2014: 428-433.
[88] Khoshgoftaar T M, Gao K, Napolitano A. Exploring an iterative feature selection technique for highly imbalanced data sets[C]//2012 IEEE 13th International Conference on Information Reuse & Integration (IRI). IEEE, 2012: 101-108.
[89] Mizuno O, Hata H. A metric to detect fault-prone software modules using text filtering[J]. International Journal of Reliability and Safety, 2013, 7(1): 17-31.
[90] Altidor W, Khoshgoftaar T M, Gao K. Wrapper-based feature ranking techniques for determining relevance of software engineering metrics[J]. International Journal of Reliability, Quality and Safety Engineering, 2010, 17(05): 425-464.
[91] Kumari D, Rajnish K. Comparing Efficiency of Software Fault Prediction Models Developed Through Binary and Multinomial Logistic Regression Techniques[M]//Information Systems Design and Intelligent Applications. Springer, New Delhi, 2015: 187-197.
[92] Wang H, Khoshgoftaar T M, Wald R. Measuring stability of feature selection techniques on real-world software datasets[M]//Information Reuse and Integration in Academia and Industry. Springer, Vienna, 2013: 113-132.
[93] Oyetoyan T D, Cruzes D S, Conradi R. Transition and defect patterns of components in dependency cycles during software evolution[C]//2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE, 2014: 283-292.
[94] Jia H, Shu F, Yang Y, et al. Data transformation and attribute subset selection: Do they help make differences in software failure prediction?[C]//2009 IEEE International Conference on Software Maintenance. IEEE, 2009: 519-522.
[95] Wang H, Khoshgoftaaar T M, Liang Q. Stability and classification performance of feature selection techniques[C]//2011 10th International Conference on Machine Learning and Applications and Workshops. IEEE, 2011, 1: 151-156.
[96] Mahmood Z, Bowes D, Hall T, et al. Reproducibility and replicability of software defect prediction studies[J]. Information and Software Technology, 2018, 99: 148-163.
[97] Mizuno O. On effects of tokens in source code to accuracy of fault-prone module prediction[C]//2013 International Computer Science and Engineering Conference (ICSEC). IEEE, 2013: 103-108.
[98] Gao K, Khoshgoftaar T M, Napolitano A. An empirical investigation of combining filter-based feature subset selection and data sampling for software defect prediction[J]. International Journal of Reliability, Quality and Safety Engineering, 2015, 22(06): 1550027.
[99] Zimmermann T, Nagappan N, Williams L, et al. An empirical study of the factors relating field failures and dependencies[C]//IEEE Fourth International Conference on Software Testing, Verification and Validation. 2011: 347-356.
[100] Gao K, Khoshgoftaar T M, Napolitano A. Aggregating data sampling with feature subset selection to address skewed software defect data[J]. International Journal of Software Engineering and Knowledge Engineering, 2015, 25(09n10): 1531-1550.
[101] Mizuno O, Hirata Y. Fault-prone module prediction using contents of comment lines[C]//Proc. of International Workshop on Empirical Software Engineering in Practice 2010 (IWESEP2010). 2010, 12: 39-44.
[102] Krishnan S. Evidence-based defect assessment and prediction for software product lines[J]. 2013.
[103] Gao K, Khoshgoftaar T M. Assessments of Feature Selection Techniques with Respect to Data Sampling for Highly Imbalanced Software

Measurement Data[J]. International Journal of Reliability, Quality and Safety Engineering, 2015, 22(02): 1550010.
[104] Muthukumaran K, Dasgupta A, Abhidnya S, et al. On the effectiveness of cost sensitive neural networks for software defect prediction[C]//International Conference on Soft Computing and Pattern Recognition. Springer, Cham, 2016: 557-570.
[105] Han W, Lung C H, Ajila S A. Empirical investigation of code and process metrics for defect prediction[C]//2016 IEEE Second International Conference on Multimedia Big Data (BigMM). IEEE, 2016: 436-439.
[106] Gao K, Khoshgoftaar T M, Napolitano A. Investigating two approaches for adding feature ranking to sampled ensemble learning for software quality estimation[J]. International Journal of Software Engineering and Knowledge Engineering, 2015, 25(01): 115-146.
[107] Kumari D, Rajnish K. Investigating the Effect of Object-oriented Metrics on Fault Proneness Using Empirical Analysis[J]. International Journal of Software Engineering and Its Applications, 2015, 9(2): 171-188.
[108] Maheshwari S, Agarwal S. Three-way decision based Defect Prediction for Object Oriented Software[C]//Proceedings of the International Conference on Advances in Information Communication Technology & Computing. ACM, 2016: 4.
[109] Kumari D, Rajnish K. Evaluating The Impact Of Binary And Multinomial LR In Developing Software Fault Prediction Model Using OO-Metrics[J]. Global Journal of Pure and Applied Mathematics, 2015, 11(1): 437-462.
[110] Bettenburg N. Studying the Impact of Developer Communication on the Quality and Evolution of a Software System[D]. , 2014.
[111] Shriram C K, Muthukumaran K, Bhanu Murthy N L. Empirical Study on the Distribution of Bugs in Software Systems[J]. International Journal of Software Engineering and Knowledge Engineering, 2018, 28(01): 97-122.
[112] Muthukumaran K, Srinivas S, Malapati A, et al. Software Defect Prediction Using Augmented Bayesian Networks[C]//International Conference on Soft Computing and Pattern Recognition. Springer, Cham, 2016: 279-293.
[113] Li H Y, Li M, Zhou Z H. Towards one reusable model for various software defect mining tasks[J].
[114] Jiarpakdee J, Tantithamthavorn C, Treude C. Artefact: An R Implementation of the AutoSpearman Function[C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2018: 711-711.
[115] Wang H, Khoshgoftaar T M, Napolitano A. Choosing the Best Classification Performance Metric for Wrapper-based Software Metric Selection for Defect Prediction[C]//SEKE. 2014: 540-545.
[116] Buchari M A, Mardiyanto S, Hendradjaya B. Implementation of Chaotic Gaussian Particle Swarm Optimization for Optimize Learning-to-Rank Software Defect Prediction Model Construction[C]//Journal of Physics: Conference Series. IOP Publishing, 2018, 978(1): 012079.
[117] Nagappan M, Murphy B, Vouk M. Which code construct metrics are symptoms of post release failures?[C]//Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics. ACM, 2011: 65-68.
[118] Herraiz Tabernero I, Shihab E, Nguyen T H D, et al. Impact of Installation Counts on Perceived Quality: A Case Study on Debian[J]. 2011.
[119] Verma R, Gupta A. An approach of attribute selection for reducing false alarms[C]//2012 CSI Sixth International Conference on Software Engineering (CONSEG). IEEE, 2012: 1-7.
[120] Rathore S S, Kumar S. An Approach for the Prediction of Number of Software Faults Based on the Dynamic Selection of Learning Techniques[J]. IEEE Transactions on Reliability, 2018 (99): 1-21.
[121] Zeller A, Zimmermann T, Bird C. Failure is a four-letter word[J]. 2011.
[122] Hashem A. A COMPREHENSIVE EMPIRICAL VALIDATION OF PACKAGE-LEVEL METRICS FOR OO SYSTEMS[D]. King Fahd University of Petroleum and Minerals, 2010.
[123] Khoshgoftaar T M, Gao K, Chen Y, et al. Comparing Feature Selection Techniques for Software Quality Estimation Using Data-Sampling-Based Boosting Algorithms[J]. International Journal of Reliability, Quality and Safety Engineering, 2015, 22(03): 1550013.
[124] Wang H, Khoshgoftaar T M, Wald R, et al. A novel dataset-similarity-aware approach for evaluating stability of software metric selection techniques[C]//2012 IEEE 13th International Conference on Information Reuse & Integration (IRI). IEEE, 2012: 1-8.
[125] Jiarpakdee J, Tantithamthavorn C, Treude C. The impact of automated feature selection techniques on the interpretation of defect models[J]. Empirical Software Engineering, 2020: 1-49.
[126] Goseva-Popstojanova K, Ahmad M, Alshehri Y. Software fault proneness prediction with group lasso regression: On factors that affect classification performance[C]//2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2019, 2: 336-343.
[127] Xu Z, Ye S, Zhang T, et al. MVSE: Effort-Aware Heterogeneous Defect Prediction via Multiple-View Spectral Embedding[C]//2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2019: 10-17.
[128] Huo X, Li M. On cost-effective software defect prediction: Classification or ranking?[J]. Neurocomputing, 2019, 363: 339-350.
[129] Pham V, Lokan C, Kasmarik K. A Better Set of Object-Oriented Design Metrics for Within-Project Defect Prediction[M]//Proceedings of the Evaluation and Assessment in Software Engineering. 2020: 230-239.
[130] Rahman A. Software Defect Prediction Using Rich Contextualized Language Use Vectors[D]. , 2019.
[131] Zheng W, Mo S, Jin X, et al. Software Defect Prediction Model Based on Improved Deep Forest and AutoEncoder by Forest[C]//SEKE. 2019: 419-540.
[132] Yang X. Evaluating Software Metrics for Sorting Software Modules in Order of Defect Count[C]//ICSOFT. 2019: 94-105.
[133] Li H Y, Li M, Zhou Z H. Towards one reusable model for various software defect mining tasks[C]//Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Cham, 2019: 212-224.
[134] Nair P R. Optimizing bug prediction in software testing using Super Learner[D]. Dublin, National College of Ireland, 2019.
[135] Bassuday K, Ahmed M. Fault Prediction in Android Systems through AI[J]. 2019.
[136] Tosun Misirli A, Murphy B, Zimmermann T, et al. An explanatory analysis on eclipse beta-release bugs through in-process metrics[C]//Proceedings of the 8th international workshop on Software quality. ACM, 2011: 26-33.
[137] Arshad A, Riaz S, Jiao L, et al. The empirical study of semi-supervised deep fuzzy C-mean clustering for software fault prediction[J]. IEEE Access, 2018, 6: 47047-47061.
[138] Kidwell B R. MiSFIT: Mining Software Fault Information and Types[J]. 2015.
[139] Shaikh M, Lee K S, Lee C G. Assessing the Bug-Prediction with Re-Usability Based Package Organization for Object Oriented Software Systems[J]. IEICE TRANSACTIONS on Information and Systems, 2017, 100(1): 107-117.
[140] Ferenc R, Tóth Z, Ladányi G, et al. A public unified bug dataset for java and its assessment regarding metrics and bug prediction[J]. Software Quality Journal, 2020: 1-60.
[141] Ferenc R, Siket I, Hegedűs P, et al. Employing Partial Least Squares Regression with Discriminant Analysis for Bug Prediction[J]. arXiv preprint arXiv:2011.01214, 2020.
[142] Wu Y, Yao J, Chang S, et al. LIMCR: Less-Informative Majorities Cleaning Rule Based on Naïve Bayes for Imbalance Learning in Software Defect Prediction[J]. Applied Sciences, 2020, 10(23): 8324.
[143] Rajbahadur G. UNDERSTANDING THE IMPACT OF EXPERIMENTAL DESIGN CHOICES ON MACHINE LEARNING CLASSIFIERS IN SOFTWARE ANALYTICS[D].
[144] Pham V, Lokan C, Kasmarik K. A Better Set of Object-Oriented Design Metrics for Within-Project Defect Prediction[M]//Proceedings of the Evaluation and Assessment in Software Engineering. 2020: 230-239.

The list of citations using the Metrics-Repo-2010 data set

[145] He Z, Shu F, Yang Y, et al. An investigation on the feasibility of cross-project defect prediction[J]. Automated Software Engineering, 2012, 19(2):

167-199.
[146] Peters F, Menzies T, Marcus A. Better cross company defect prediction[C]//Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013: 409-418.
[147] He P, Li B, Liu X, et al. An empirical study on software defect prediction with a simplified metric set[J]. Information and Software Technology, 2015, 59: 170-190.
[148] Madeyski L, Jureczko M. Which process metrics can significantly improve defect prediction models? An empirical study[J]. Software Quality Journal, 2015, 23(3): 393-422.
[149] Turhan B, Mısırlı A T, Bener A. Empirical evaluation of the effects of mixed project data on learning defect predictors[J]. Information and Software Technology, 2013, 55(6): 1101-1118.
[150] Chen L, Fang B, Shang Z, et al. Negative samples reduction in cross-company software defects prediction[J]. Information and Software Technology, 2015, 62: 67-77.
[151] Jureczko M. Significance of different software metrics in defect prediction[J]. Software Engineering: An International Journal, 2011, 1(1): 86-95.
[152] Bennin K E, Keung J, Phannachitta P, et al. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction[J]. IEEE Transactions on Software Engineering, 2018, 44(6): 534-550.
[153] Ryu D, Jang J I, Baik J. A transfer cost-sensitive boosting approach for cross-project defect prediction[J]. Software Quality Journal, 2017, 25(1): 235-272.
[154] Shatnawi R. Deriving metrics thresholds using log transformation[J]. Journal of Software: Evolution and Process, 2015, 27(2): 95-113.
[155] Alenezi M, Magel K. Empirical evaluation of a new coupling metric: Combining structural and semantic coupling[J]. International Journal of Computers and Applications, 2014, 36(1): 34-44.
[156] Hosseini S, Turhan B, Mäntylä M. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction[J]. Information and Software Technology, 2018, 95: 296-312.
[157] Yu L. Using negative binomial regression analysis to predict software faults: A study of apache ant[J]. 2012.
[158] Mizuno O, Hirata Y. A cross-project evaluation of text-based fault-prone module prediction[C]//2014 6th International Workshop on Empirical Software Engineering in Practice. IEEE, 2014: 43-48.
[159] Kaur A, Kaur K. Performance analysis of ensemble learning for predicting defects in open source software[C]//2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2014: 219-225.
[160] Hosseini S, Turhan B, Mäntylä M. Search based training data selection for cross project defect prediction[C]//Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering. ACM, 2016: 3.
[161] Lumpe M, Vasa R, Menzies T, et al. Learning better inspection optimization policies[J]. International Journal of Software Engineering and Knowledge Engineering, 2012, 22(05): 621-644.
[162] Shatnawi R. The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction[J]. Innovations in Systems and Software Engineering, 2017, 13(2-3): 201-217.
[163] Alenezi M, Zarour M. Modularity measurement and evolution in object-oriented open-source projects[C]//Proceedings of the The International Conference on Engineering & MIS 2015. ACM, 2015: 16.
[164] Poon W N, Bennin K E, Huang J, et al. Cross-project defect prediction using a credibility theory based naive bayes classifier[C]//2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2017: 434-441.
[165] Xu Z, Liu J, Luo X, et al. Cross-version defect prediction via hybrid active learning with kernel principal component analysis[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 209-220.
[166] Xu Z, Liu J, Luo X, et al. Software defect prediction based on kernel PCA and weighted extreme learning machine[J]. Information and Software Technology, 2019, 106: 182-200.
[167] Bennin K E, Keung J, Monden A. Impact of the distribution parameter of data sampling approaches on software defect prediction models[C]//2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2017: 630-635.
[168] Bennin K E, Keung J W, Monden A. On the relative value of data resampling approaches for software defect prediction[J]. Empirical Software Engineering, 2019, 24(2): 602-636.
[169] Mizuno O. On effects of tokens in source code to accuracy of fault-prone module prediction[C]//2013 International Computer Science and Engineering Conference (ICSEC). IEEE, 2013: 103-108.
[170] Bluemke I, Stepień A. Experiment on defect prediction[C]//International Conference on Dependability and Complex Systems. Springer, Cham, 2015: 25-34.
[171] Gupta S, Gupta D L. Fault Prediction using Metric Threshold Value of Object Oriented Systems[J]. International Journal of Engineering Science, 2017, 13629.
[172] Shatnawi R. Synergies and conflicts among software quality attributes and bug fixes[J]. International Journal of Information Systems and Change Management, 2017, 9(1): 3-21.
[173] Yang Y. Software Defect Prediction Model Research for Network and Cloud Software Development[C]//2017 5th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2017). Atlantis Press, 2017.
[174] Shatnawi R. Improving Software Fault Prediction With Threshold Values[C]//2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM). IEEE, 2018: 1-6.
[175] Yuand Y T, Zhange T. Software Defect Prediction Based on Kernel PCA and Weighted Extreme Learning Machine[J].
[176] Bispo A, Prudêncio R, Véras D. Instance Selection and Class Balancing Techniques for Cross Project Defect Prediction[C]//2018 7th Brazilian Conference on Intelligent Systems (BRACIS). IEEE, 2018: 552-557.
[177] Ryu D, Baik J. A Comparative Study on Similarity Measure Techniques for Cross-Project Defect Prediction[J]. KIPS Transactions on Software and Data Engineering, 2018, 7(6): 205-220.
[178] Watanabe T, Monden A, Yücel Z, et al. Cross-validation-based association rule prioritization metric for software defect characterization[J]. IEICE TRANSACTIONS on Information and Systems, 2018, 101(9): 2269-2278.
[179] Malhotra R, Bansal A J. Fault prediction considering threshold effects of object-oriented metrics[J]. Expert Systems, 2015, 32(2): 203-219.
[180] Bennin K E, Keung J, Monden A, et al. The significant effects of data sampling approaches on software defect prioritization and classification[C]//Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE Press, 2017: 364-373.
[181] Alenezi M, Banitaan S, Obeidat Q. Fault-proneness of open source systems: An empirical analysis[J]. Synapse, 2014, 1: 256.
[182] Alenezi M, Zarour M. Does software structures quality improve over software evolution? Evidences from open-source projects[J]. International Journal of Computer Science and Information Security, 2016, 14: 61.
[183] He P, He Y, Yu L, et al. An Improved Method for Cross-Project Defect Prediction by Simplifying Training Data[J]. Mathematical Problems in Engineering, 2018, 2018.
[184] Jureczko M, Magott J. QualitySpy: a framework for monitoring software development processes[J]. Journal of Theoretical and Applied Computer Science, 2012, 6(1): 35-45.
[185] Jureczko M, Madeyski L. Cross–project defect prediction with respect to code ownership model: An empirical study[J]. e-Informatica Software Engineering Journal, 2015, 9(1).
[186] Shatnawi R. Empirical study of fault prediction for open-source systems using the Chidamber and Kemerer metrics[J]. IET software, 2013, 8(3):

113-119.
[187] Haouari A T, Souici-Meslati L, Atil F, et al. Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction[J]. Applied Soft Computing, 2020: 106686.
[188] Tsoukalas D, Kehagias D, Siavvas M, et al. Technical debt forecasting: An empirical study on open-source repositories[J]. Journal of Systems and Software, 2020: 110777.
[189] Shao Y, Liu B, Wang S, et al. Software defect prediction based on correlation weighted class association rule mining[J]. Knowledge-Based Systems, 2020: 105742.
[190] Bal P R, Kumar S. WR-ELM: Weighted Regularization Extreme Learning Machine for Imbalance Learning in Software Fault Prediction[J]. IEEE Transactions on Reliability, 2020.
[191] Tong H, Liu B, Wang S, et al. Transfer-learning oriented class imbalance learning for cross-project defect prediction[J]. arXiv preprint arXiv:1901.08429, 2019.
[192] Alqmase M, Alshayeb M, Ghouti L. Threshold Extraction Framework for Software Metrics[J]. Journal of Computer Science and Technology, 2019, 34(5): 1063-1078.
[193] Xu Z, Li S, Xu J, et al. LDFR: Learning deep feature representation for software defect prediction[J]. Journal of Systems and Software, 2019, 158: 110402.
[194] Hosseini S. ASCIENTIAE RERUM[J].
[195] Sun Z, Li J, Sun H. An empirical study of public data quality problems in cross project defect prediction[J]. arXiv preprint arXiv:1805.10787, 2018.
[196] He P, Li B, Ma Y. Towards cross-project defect prediction with imbalanced feature sets[J]. arXiv preprint arXiv:1411.4228, 2014.
[197] Tantithamthavorn C, McIntosh S, Hassan A E, et al. Automated parameter optimization of classification techniques for defect prediction models[C]//Proceedings of the 38th International Conference on Software Engineering. 2016: 321-332.
[198] Tantithamthavorn C, McIntosh S, Hassan A E, et al. An empirical comparison of model validation techniques for defect prediction models[J]. IEEE Transactions on Software Engineering, 2016, 43(1): 1-18.
[199] Xia X, Lo D, Pan S J, et al. Hydra: Massively compositional model for cross-project defect prediction[J]. IEEE Transactions on software Engineering, 2016, 42(10): 977-998.
[200] Zhang F, Zheng Q, Zou Y, et al. Cross-project defect prediction using a connectivity-based unsupervised classifier[C]//2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 2016: 309-320.
[201] Herbold S. Training data selection for cross-project defect prediction[C]//Proceedings of the 9th international conference on predictive models in software engineering. 2013: 1-10.
[202] Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network[C]//2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2017: 318-328.
[203] He Z, Peters F, Menzies T, et al. Learning from open-source projects: An empirical study on defect prediction[C]//2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013: 45-54.
[204] Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches[J]. IEEE Transactions on Software Engineering, 2017, 44(9): 811-833.
[205] Tantithamthavorn C, McIntosh S, Hassan A E, et al. The impact of automated parameter optimization on defect prediction models[J]. IEEE Transactions on Software Engineering, 2018, 45(7): 683-711.
[206] Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction[J]. IEEE Transactions on Software Engineering, 2017, 45(2): 111-147.
[207] Peters F, Menzies T, Layman L. LACE2: Better privacy-preserving data sharing for cross project defect prediction[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 801-811.
[208] Bowes D, Hall T, Petrić J. Software defect prediction: do different classifiers find the same defects?[J]. Software Quality Journal, 2018, 26(2): 525-552.
[209] Krishna R, Menzies T, Fu W. Too much automation? The bellwether effect and its implications for transfer learning[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 122-131.
[210] Ryu D, Baik J. Effective multi-objective naïve Bayes learning for cross-project defect prediction[J]. Applied Soft Computing, 2016, 49: 1062-1077.
[211] Palomba F, Zanoni M, Fontana F A, et al. Toward a smell-aware bug prediction model[J]. IEEE Transactions on Software Engineering, 2017, 45(2): 194-218.
[212] Arar Ö F, Ayan K. Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies[J]. Expert Systems with Applications, 2016, 61: 106-121.
[213] Herbold S, Trautsch A, Grabowski J. Global vs. local models for cross-project defect prediction[J]. Empirical software engineering, 2017, 22(4): 1866-1902.
[214] Zhou Y, Yang Y, Lu H, et al. How far we have progressed in the journey? an examination of cross-project defect prediction[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2018, 27(1): 1-51.
[215] Krishna R, Menzies T. Bellwethers: A baseline method for transfer learning[J]. IEEE Transactions on Software Engineering, 2018, 45(11): 1081-1105.
[216] Kawata K, Amasaki S, Yokogawa T. Improving relevancy filter methods for cross-project defect prediction[C]//2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence. IEEE, 2015: 2-7.
[217] Ma W, Chen L, Yang Y, et al. Empirical analysis of network measures for effort-aware fault-proneness prediction[J]. Information and Software Technology, 2016, 69: 50-70.
[218] Zhang F, Keivanloo I, Zou Y. Data transformation in cross-project defect prediction[J]. Empirical Software Engineering, 2017, 22(6): 3186-3218.
[219] Amasaki S, Kawata K, Yokogawa T. Improving cross-project defect prediction methods with data simplification[C]//2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE, 2015: 96-103.
[220] Yan M, Fang Y, Lo D, et al. File-level defect prediction: Unsupervised vs. supervised models[C]//2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2017: 344-353.
[221] Turabieh H, Mafarja M, Li X. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction[J]. Expert systems with applications, 2019, 122: 27-42.
[222] Boucher A, Badri M. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison[J]. Information and Software Technology, 2018, 96: 38-67.
[223] Huang J, Keung J W, Sarro F, et al. Cross-validation based K nearest neighbor imputation for software quality datasets: An empirical study[J]. Journal of Systems and Software, 2017, 132: 226-252.
[224] Li Z, Jing X Y, Zhu X, et al. Heterogeneous defect prediction through multiple kernel learning and ensemble learning[C]//2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017: 91-102.
[225] Liu C, Yang D, Xia X, et al. A two-phase transfer learning model for cross-project defect prediction[J]. Information and Software Technology, 2019, 107: 125-136.
[226] Chen X, Zhang D, Zhao Y, et al. Software defect number prediction: Unsupervised vs supervised methods[J]. Information and Software Technology, 2019, 106: 161-181.

[227] Krishna R, Menzies T, Layman L. Less is more: Minimizing code reorganization using XTREE[J]. Information and Software Technology, 2017, 88: 53-66.
[228] Chen X, Shen Y, Cui Z, et al. Applying feature selection to software defect prediction using multi-objective optimization[C]//2017 IEEE 41st annual computer software and applications conference (COMPSAC). IEEE, 2017, 2: 54-59.
[229] Herbold S. CrossPare: A tool for benchmarking cross-project defect predictions[C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE, 2015: 90-96.
[230] Boucher A, Badri M. Using software metrics thresholds to predict fault-prone classes in object-oriented software[C]//2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD). IEEE, 2016: 169-176.
[231] Xiaoxing Yang, Wushao Wen. Ridge and Lasso Regression Models for Cross-Version Defect Prediction. IEEE Trans. Reliab. 67(3): 885-896 (2018).
[232] Zhang Y, Lo D, Xia X, et al. Combined classifier for cross-project defect prediction: an extended empirical study[J]. Frontiers of Computer Science, 2018, 12(2): 280-296.
[233] Herbold S. A systematic mapping study on cross-project defect prediction[J]. arXiv preprint arXiv:1705.06429, 2017.
[234] He P, Li B, Zhang D, et al. Simplification of training data for cross-project defect prediction[J]. arXiv preprint arXiv:1405.0773, 2014.
[235] Li Y, Huang Z, Wang Y, et al. Evaluating data filter on cross-project defect prediction: Comparison and improvements[J]. IEEE Access, 2017, 5: 25646-25656.
[236] Zhou Xu, Shuai Li, Yutian Tang, Xiapu Luo, Tao Zhang, Jin Liu, Jun Xu. Cross version defect prediction with representative data via sparse subset selection. ICPC 2018: 132-143.
[237] Ni C, Chen X, Wu F, et al. An empirical study on pareto based multi-objective feature selection for software defect prediction[J]. Journal of Systems and Software, 2019, 152: 215-238.
[238] Kondo M, Bezemer C P, Kamei Y, et al. The impact of feature reduction techniques on defect prediction models[J]. Empirical Software Engineering, 2019, 24(4): 1925-1963.
[239] Hussain S, Keung J, Khan A A, et al. Performance evaluation of ensemble methods for software fault prediction: An experiment[C]//Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference. 2015: 91-95.
[240] Boucher A, Badri M. Predicting fault-prone classes in object-oriented software: an adaptation of an unsupervised hybrid SOM algorithm[C]//2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2017: 306-317.
[241] Hussain S, Keung J, Khan A A, et al. Detection of fault-prone classes using logistic regression based object-oriented metrics thresholds[C]//2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2016: 93-100.
[242] Porto F, Minku L, Mendes E, et al. A systematic study of cross-project defect prediction with meta-learning[J]. arXiv preprint arXiv:1802.06025, 2018.
[243] Zhang X, Ben K, Zeng J. Cross-entropy: A new metric for software defect prediction[C]//2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2018: 111-122.
[244] Herbold S. Comments on ScottKnottESD in response to" An empirical comparison of model validation techniques for defect prediction models"[J]. IEEE Transactions on Software Engineering, 2017, 43(11): 1091-1094.
[245] Chen D, Fu W, Krishna R, et al. Applications of psychological science for actionable analytics[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 456-467.
[246] Tang H, Lan T, Hao D, et al. Enhancing defect prediction with static defect analysis[C]//Proceedings of the 7th Asia-Pacific Symposium on Internetware. 2015: 43-51.
[247] Li Z, Jing X Y, Zhu X. Heterogeneous fault prediction with cost-sensitive domain adaptation[J]. Software Testing, Verification and Reliability, 2018, 28(2): e1658.
[248] Liu Y, Li Y, Guo J, et al. Connecting software metrics across versions to predict defects[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 232-243.
[249] Qiu S, Lu L, Jiang S. Multiple-components weights model for cross-project software defect prediction[J]. IET Software, 2018, 12(4): 345-355.
[250] Yang J, Qian H. Defect prediction on unlabeled datasets by using unsupervised clustering[C]//2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2016: 465-472.
[251] Fan G, Diao X, Yu H, et al. Software Defect Prediction via Attention-Based Recurrent Neural Network[J]. Scientific Programming, 2019, 2019.
[252] Zhou T, Sun X, Xia X, et al. Improving defect prediction with deep forest[J]. Information and Software Technology, 2019, 114: 204-216.
[253] Prateek S, Pasala A, Aracena L M. Evaluating performance of network metrics for bug prediction in software[C]//2013 20th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2013, 1: 124-131.
[254] Sousuke Amasaki. On Applicability of Cross-project Defect Prediction Method for Multi-Versions Projects. PROMISE 2017: 93-96.
[255] Morasca S, Lavazza L. Risk-averse slope-based thresholds: Definition and empirical evaluation[J]. Information and Software Technology, 2017, 89: 37-63.
[256] Mutlu B, Sezer E A, Akcayol M A. End-to-End hierarchical fuzzy inference solution[C]//2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, 2018: 1-9.
[257] Sousuke Amasaki. Cross-Version Defect Prediction using Cross-Project Defect Prediction Approaches: Does it work? PROMISE 2018: 32-41.
[258] Goyal R, Chandra P, Singh Y. Fuzzy inferencing to identify degree of interaction in the development of fault prediction models[J]. Journal of King Saud University-Computer and Information Sciences, 2017, 29(1): 93-102.
[259] Chen J, Hu K, Yang Y, et al. Collective transfer learning for defect prediction[J]. Neurocomputing, 2019.
[260] Li Z, Jing X Y, Zhu X, et al. Heterogeneous defect prediction with two-stage ensemble learning[J]. Automated Software Engineering, 2019, 26(3): 599-651.
[261] Porto F R, Simao A. Feature Subset Selection and Instance Filtering for Cross-project Defect Prediction-Classification and Ranking[J]. CLEI Electron. J., 2016, 19(3): 4.
[262] Herbold S. Benchmarking cross-project defect prediction approaches with costs metrics[J]. arXiv preprint arXiv:1801.04107, 2018.
[263] Pan C, Lu M, Xu B, et al. An Improved CNN Model for Within-Project Software Defect Prediction[J]. Applied Sciences, 2019, 9(10): 2138.
[264] Wen W, Zhang B, Gu X, et al. An empirical study on combining source selection and transfer learning for cross-project defect prediction[C]//2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF). IEEE, 2019: 29-38.
[265] Catolino G, Palomba F, Fontana F A, et al. Improving change prediction models with code smell-related information[J]. Empirical Software Engineering, 2020, 25(1): 49-95.
[266] Cheikhi L, Abran A. An Analysis of the PROMISE and ISBSG Software Engineering Data Repositories[J]. Int. Journal of Computers and Technology, 2014, 13(5).
[267] Shriram C K, Muthukumaran K, Bhanu Murthy N L. Empirical study on the distribution of bugs in software systems[J]. International Journal of Software Engineering and Knowledge Engineering, 2018, 28(01): 97-122.
[268] Zhang X, Ben K, Zeng J. Using Cross-Entropy Value of Code for Better Defect Prediction[J]. International Journal of Performability Engineering, 2018, 14(9).
[269] Alenezi M, Abunadi I. Quality of open source systems from product metrics perspective[J]. arXiv preprint arXiv:1511.03194, 2015.
[270] Sohan M F, Kabir M A, Jabiullah M I, et al. Revisiting the class imbalance issue in software defect prediction[C]//2019 International Conference

on Electrical, Computer and Communication Engineering (ECCE). IEEE, 2019: 1-6.
[271] Li K, Xiang Z, Chen T, et al. Understanding the Automated Parameter Optimization on Transfer Learning for CPDP: An Empirical Study[J]. arXiv preprint arXiv:2002.03148, 2020.
[272] Yucalar F, Ozcift A, Borandag E, et al. Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability[J]. Engineering Science and Technology, an International Journal, 2020, 23(4): 938-950.
[273] Chen D, Chen X, Li H, et al. Deepcpdp: Deep learning based cross-project defect prediction[J]. IEEE Access, 2019, 7: 184832-184848.
[274] Krishna R, Menzies T. Actionable= Cluster+ Contrast?[C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE, 2015: 14-17.
[275] Hosseini S, Turhan B. An exploratory study of search based training data selection for cross project defect prediction[C]//2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2018: 244-251.
[276] Muthukumaran K, Dasgupta A, Abhidnya S, et al. On the effectiveness of cost sensitive neural networks for software defect prediction[C]//International Conference on Soft Computing and Pattern Recognition. Springer, Cham, 2016: 557-570.
[277] Zhang D, Chen X, Cui Z, et al. Software defect prediction model sharing under differential privacy[C]//2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE, 2018: 1547-1554.
[278] Chen X, Mu Y, Qu Y, et al. Do different cross-project defect prediction methods identify the same defective modules?[J]. Journal of Software: Evolution and Process, 2020, 32(5): e2234.
[279] Kaur I, Bajpai N. An Empirical Study on Fault Prediction using Token-Based Approach[C]//Proceedings of the International Conference on Advances in Information Communication Technology & Computing. 2016: 1-7.
[280] Huo X, Yang Y, Li M, et al. Learning Semantic Features for Software Defect Prediction by Code Comments Embedding[C]//2018 IEEE International Conference on Data Mining (ICDM). IEEE, 2018: 1049-1054.
[281] Ghosh S, Rana A, Kansal V. A Hybrid Nonlinear Manifold Detection Approach for Software Defect Prediction[C]//2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). IEEE, 2018: 453-459.
[282] Rosli M M, Tempero E, Luxton-Reilly A. What is in our datasets? Describing a structure of datasets[C]//Proceedings of the Australasian Computer Science Week Multiconference. 2016: 1-10.
[283] Wu Y, Huang S, Ji H. An Information Flow-based Feature Selection Method for Cross-Project Defect Prediction[J]. International Journal of Performability Engineering, 2018, 14(6).
[284] Di Nucci D, De Lucia A. The role of meta-learners in the adaptive selection of classifiers[C]//2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE). IEEE, 2018: 7-12.
[285] Tumar I, Hassouneh Y, Turabieh H, et al. Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction[J]. IEEE Access, 2020, 8: 8041-8055.
[286] Malhotra R. An extensive analysis of search-based techniques for predicting defective classes[J]. Computers & Electrical Engineering, 2018, 71: 611-626.
[287] Chen X, Zhang D, Cui Z Q, et al. Dp-share: Privacy-preserving software defect prediction model sharing through differential privacy[J]. Journal of Computer Science and Technology, 2019, 34(5): 1020-1038.
[288] Sohan M F, Jabiullah M I, Rahman S S M M, et al. Assessing the Effect of Imbalanced Learning on Cross-project Software Defect Prediction[C]//2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2019: 1-6.
[289] Sundström A. Investigation into predicting unit test failure using syntactic source code features[J]. 2018.
[290] Aziz S R, Khan T, Nadeem A. Experimental Validation of Inheritance Metrics' Impact on Software Fault Prediction[J]. IEEE Access, 2019, 7: 85262-85275.
[291] Ghosh S, Rana A, Kansal V. Evaluating the Impact of Sampling-Based Nonlinear Manifold Detection Model on Software Defect Prediction Problem[M]//Smart Intelligent Computing and Applications. Springer, Singapore, 2020: 141-152.
[292] Krishna R, Menzies T. Learning actionable analytics from multiple software projects[J]. Empirical Software Engineering, 2020: 1-33.
[293] Yang Y, Yang J, Qian H. Defect prediction by using cluster ensembles[C]//2018 Tenth International Conference on Advanced Computational Intelligence (ICACI). IEEE, 2018: 631-636.
[294] Jiarpakdee J, Tantithamthavorn C, Treude C. The impact of automated feature selection techniques on the interpretation of defect models[J]. Empirical Software Engineering, 2020: 1-49.
[295] Raukas H. Some approaches for software defect prediction[D]. Bachelor's Thesis (9ECTS), Institute of Computer Science Computer Science Curriculum, University of TARTU, 2017.
[296] Kaur K. Statistical Comparison of Machine Learning Techniques for Predicting Software Maintainability and Defects[J]. Guru Gobind Singh Indraprastha University, 2016.
[297] Okutan A. Use of Source Code Similarity Metrics in Software Defect Prediction[J]. arXiv preprint arXiv:1808.10033, 2018.
[298] Rizwan M, Nadeem A, Sindhu M A. Empirical Evaluation of Coupling Metrics in Software Fault Prediction[C]//2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE, 2020: 434-440.
[299] Li K, Xiang Z, Chen T, et al. BiLO-CPDP: Bi-Level Programming for Automated Model Discovery in Cross-Project Defect Prediction[J]. arXiv preprint arXiv:2008.13489, 2020.
[300] Chen X, Mu Y, Ni C, et al. Revisiting Heterogeneous Defect Prediction: How Far Are We?[J]. arXiv preprint arXiv:1908.06560, 2019.
[301] Peng K, Menzies T. How to Improve AI Tools (by Adding in SE Knowledge): Experiments with the TimeLIME Defect Reduction Tool[J]. arXiv preprint arXiv:2003.06887, 2020.
[302] Yuan Z, Chen X, Cui Z, et al. ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost[J]. IEEE Access, 2020, 8: 30037-30049.
[303] Li K, Xiang Z, Chen T, et al. Understanding the Automated Parameter Optimization on Transfer Learning for Cross-Project Defect Prediction: An Empirical Study[J].
[304] Sohan M F, Kabir M A, Rahman M, et al. Prevalence of Machine Learning Techniques in Software Defect Prediction[C]//International Conference on Cyber Security and Computer Science. Springer, Cham, 2020: 257-269.
[305] Krishnaa R, Menziesa T, Laymanb L. Recommendations for Intelligent Code Reorganization[J]. CoRR, 2016.
[306] Webster A. A Comparison of Transfer Learning Algorithms for Defect and Vulnerability Detection[R]. 2017.
[307] Amasaki S. Cross-version defect prediction: use historical data, cross-project data, or both?[J]. Empirical Software Engineering, 2020: 1-23.
[308] Deng J, Lu L, Qiu S, et al. A Suitable AST Node Granularity and Multi-Kernel Transfer Convolutional Neural Network for Cross-Project Defect Prediction[J]. IEEE Access, 2020, 8: 66647-66661.
[309] Ouellet A, Badri M. Empirical Analysis of Object-Oriented Metrics and Centrality Measures for Predicting Fault-Prone Classes in Object-Oriented Software[C]//International Conference on the Quality of Information and Communications Technology. Springer, Cham, 2019: 129-143.
[310] Qu Y, Liu T, Chi J, et al. node2defect: using network embedding to improve software defect prediction[C]//2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2018: 844-849.
[311] Ha D A, Chen T H, Yuan S M. Unsupervised methods for Software Defect Prediction[C]//Proceedings of the Tenth International Symposium on Information and Communication Technology. 2019: 49-55.
[312] Moudache S, Badri M. Software Fault Prediction Based on Fault Probability and Impact[C]//2019 18th IEEE International Conference On

Machine Learning And Applications (ICMLA). IEEE, 2019: 1178-1185.
[313] Ren J H, Liu F. Predicting Software Defects Using Self-Organizing Data Mining[J]. IEEE Access, 2019, 7: 122796-122810.
[314] Ahmed M R, Ali M A, Ahmed N, et al. The Impact of Software Fault Prediction in Real-World Application: An Automated Approach for Software Engineering[C]//Proceedings of 2020 the 6th International Conference on Computing and Data Engineering. 2020: 247-251.
[315] Bangash A A, Sahar H, Hindle A, et al. On the Time-Based Conclusion Stability of Software Defect Prediction Models[J]. arXiv preprint arXiv: 1911.06348v3, 2019.
[316] Sohan M F, Kabir M A, Rahman M, et al. Training Data Selection Using Ensemble Dataset Approach for Software Defect Prediction[C]//International Conference on Cyber Security and Computer Science. Springer, Cham, 2020: 243-256.
[317] Krishna Prasad R. Learning Actionable Analytics in Software Engineering[J]. 2019.
[318] Peng K, Menzies T. Defect Reduction Planning (using TimeLIME)[J]. arXiv preprint arXiv:2006.07416, 2020.
[319] Kumar S, Rathore S S. Evaluation of Techniques for Binary Class Classification[M]//Software Fault Prediction. Springer, Singapore, 2018: 39-57.
[320] Krishna R, Menzies T. From Prediction to Planning: Improving Software Quality with BELLTREE[J].
[321] Li H, Li X, Chen X, et al. Cross-project Defect Prediction via ASTToken2Vec and BLSTM-based Neural Network[C]//2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019: 1-8.
[322] Zhang J, Wu J, Chen C, et al. CDS: A Cross–Version Software Defect Prediction Model With Data Selection[J]. IEEE Access, 2020, 8: 110059-110072.
[323] Kumar K, Jayadev Gyani D, Narsimha G. Software Defect Prediction using Ant Colony Optimization[J]. International Journal of Applied Engineering Research, 2018, 13(19): 14291-14297.
[324] Morasca S, Lavazza L. On the assessment of software defect prediction models via ROC curves[J]. Empirical Software Engineering, 2020: 1-43.
[325] Zhang Q, Wu B. Software Defect Prediction via Transformer[C]//2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE, 2020, 1: 874-879.
[326] Hussain S, Khan A A, Bennin K E. Empirical investigation of fault predictors in context of class membership probability estimation[C]//Proceedings of the 31st Annual ACM Symposium on Applied Computing. 2016: 1550-1553.
[327] Gao H, Lu M, Pan C, et al. Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction?[C]//2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2019: 1-5.
[328] Mutlu B, Sezer E A, Akcayol M A. Automatic Rule Generation of Fuzzy Systems: A Comparative Assessment on Software Defect Prediction[C]//2018 3rd International Conference on Computer Science and Engineering (UBMK). IEEE, 2018: 209-214.
[329] De Lucia A, Salza P. Parallel Genetic Algorithms in the Cloud[J]. 2017.
[330] Catal C, Erdogan M, Isik C. Software Defect Prediction in the Cloud[J].
[331] Srivastava K. Cohesion Based Testability Model: A New Perspective[J].
[332] Fujiwara T, Mizuno O, Leelaprute P. Fault-Prone Byte-Code Detection Using Text Classifier[C]//International Conference on Product-Focused Software Process Improvement. Springer, Cham, 2015: 415-430.
[333] Fan G, Diao X, Yu H, et al. Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction[C]//2019 26th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2019: 244-251.
[334] AGRAWAL A. CROSS PROJECT DEFECT PREDICTION[D]. , 2018.
[335] Sara E, Laila C, Ali I. The Impact of SMOTE and Grid Search on Maintainability Prediction Models[C]//2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2019: 1-8.
[336] Chen H, Jing X Y, Li Z, et al. An Empirical Study on Heterogeneous Defect Prediction Approaches[J]. IEEE Transactions on Software Engineering, 2020.
[337] Ryu D, Baik J. Effective Harmony Search-Based Optimization of Cost-Sensitive Boosting for Improving the Performance of Cross-Project Defect Prediction[J]. KIPS Transactions on Software and Data Engineering, 2018, 7(3): 77-90.
[338] Rai P, Kumar S, Verma D K. Prediction of Software Effort Using Design Metrics: An Empirical Investigation[M]//Social Networking and Computational Intelligence. Springer, Singapore, 2020: 627-637.
[339] Bangash A A, Sahar H, Hindle A, et al. On the time-based conclusion stability of cross-project defect prediction models[J]. Empirical Software Engineering, 2020: 1-38.
[340] Yao Z, Song J, Liu Y, et al. Research on Cross-version Software Defect Prediction Based on Evolutionary Information[C]//IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2019, 563(5): 052092.
[341] Kabir M A, Keung J W, Bennin K E, et al. A Drift Propensity Detection Technique to Improve the Performance for Cross-Version Software Defect Prediction[C]//2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2020: 882-891.
[342] Muthukumaran K, Murthy N L B, Janani P S. Empirical Study on the Distribution of Object-Oriented Metrics in Software Systems[C]//International Conference on Information and Software Technologies. Springer, Cham, 2019: 299-317.
[343] Peters F. LACE: Supporting Privacy-Preserving Data Sharing in Transfer Defect Learning[J]. 2014.
[344] Li J, Jing X Y, Wu F, et al. A Cost-Sensitive Shared Hidden Layer Autoencoder for Cross-Project Defect Prediction[C]//Chinese Conference on Pattern Recognition and Computer Vision (PRCV). Springer, Cham, 2019: 491-502.
[345] Di Nucci D. Methods and tools for focusing and prioritizing the testing effort[C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2018: 722-726.
[346] Sun Y, Jing X Y, Wu F, et al. Adversarial Learning for Cross-Project Semi-Supervised Defect Prediction[J]. IEEE Access, 2020, 8: 32674-32687.
[347] Rathore S S, Kumar S. Homogeneous Ensemble Methods for the Prediction of Number of Faults[M]//Fault Prediction Modeling for the Prediction of Number of Software Faults. Springer, Singapore, 2019: 31-45.
[348] Tahir A, Bennin K E, MacDonell S G, et al. Revisiting the size effect in software fault prediction models[C]//Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2018: 1-10.
[349] Gong L, Jiang S, Jiang L. Conditional Domain Adversarial Adaptation for Heterogeneous Defect Prediction[J]. IEEE Access, 2020, 8: 150738-150749.
[350] Jiang K, Zhang Y, Wu H, et al. Heterogeneous defect prediction based on transfer learning to handle extreme imbalance[J]. Applied Sciences, 2020, 10(1): 396.
[351] Kaen E, Algarni A. Feature Selection Approach for Improving the Accuracy of Software Bug Prediction[J].
[352] Huo X, Li M. On cost-effective software defect prediction: Classification or ranking?[J]. Neurocomputing, 2019, 363: 339-350.
[353] Ren J, Liu F. A Novel Approach for Software Defect prediction Based on the Power Law Function[J]. Applied Sciences, 2020, 10(5): 1892.
[354] Maruf O M. The impact of parameter optimization of ensemble learning on defect prediction[J]. Computer Science Journal of Moldova, 2019, 79(1): 85-128.
[355] Xiaoxing Yang, Xin Li, Wushao Wen, Jianmin Su. An Investigation of Ensemble Approaches to Cross-Version Defect Prediction. SEKE 2019: 437-556.
[356] Peters F, Menzies T, Gong L, et al. Balancing privacy and utility in cross-company defect prediction[J]. IEEE Transactions on Software Engineering, 2013, 39(8): 1054-1068.
[357] Sarro F, Di Martino S, Ferrucci F, et al. A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction[C]//Proceedings of the 27th annual ACM symposium on applied computing. 2012: 1215-1220.
[358] Rathore S S, Kumar S. Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems[J]. Knowledge-

Based Systems, 2017, 119: 232-256.
[359] Stuckman J, Walden J, Scandariato R. The effect of dimensionality reduction on software vulnerability prediction models[J]. IEEE Transactions on Reliability, 2016, 66(1): 17-37.
[360] Erturk E, Sezer E A. Iterative software fault prediction with a hybrid approach[J]. Applied Soft Computing, 2016, 49: 1020-1033.
[361] Foucault M, Falleri J R, Blanc X. Code ownership in open-source software[C]//Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. 2014: 1-9.
[362] Gupta D L, Saxena K. Software bug prediction using object-oriented metrics[J]. Sādhanā, 2017, 42(5): 655-669.
[363] Shukla S, Radhakrishnan T, Muthukumaran K, et al. Multi-objective cross-version defect prediction[J]. Soft Computing, 2018, 22(6): 1959-1980.
[364] Xuan X, Lo D, Xia X, et al. Evaluating defect prediction approaches using a massive set of metrics: An empirical study[C]//Proceedings of the 30th Annual ACM Symposium on Applied Computing. 2015: 1644-1647.
[365] Qing H, Biwen L, Beijun S, et al. Cross-project software defect prediction using feature-based transfer learning[C]//Proceedings of the 7th Asia-Pacific Symposium on Internetware. 2015: 74-82.
[366] Ferenc R, Tóth Z, Ladányi G, et al. A public unified bug dataset for Java[C]//Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering. 2018: 12-21.
[367] Kaur A, Kaur K. Value and applicability of academic projects defect datasets in cross-project software defect prediction[C]//2016 2nd International Conference on Computational Intelligence and Networks (CINE). IEEE, 2016: 154-159.
[368] Gong L, Jiang S, Bo L, et al. A novel class-imbalance learning approach for both within-project and cross-project defect prediction[J]. IEEE Transactions on Reliability, 2019, 69(1): 40-54.
[369] Wang F, Huang J, Ma Y. A Top-k Learning to Rank Approach to Cross-Project Software Defect Prediction[C]//2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2018: 335-344.
[370] Du Y, Zhang L, Shi J, et al. Feature-grouping-based two steps feature selection algorithm in software defect prediction[C]//Proceedings of the 2nd International Conference on Advances in Image Processing. 2018: 173-178.
[371] Qiu S, Lu L, Cai Z, et al. Cross-Project Defect Prediction via Transferable Deep Learning-Generated and Handcrafted Features[C]//SEKE. 2019: 431-552.
[372] Goel L, Gupta S. Cross Projects Defect Prediction Modeling[M]//Data Visualization and Knowledge Engineering. Springer, Cham, 2020: 1-21.
[373] Cai Z, Lu L, Qiu S. An abstract syntax tree encoding method for cross-project defect prediction[J]. IEEE Access, 2019, 7: 170844-170853.
[374] Yu Q, Jiang S, Qian J, et al. Process metrics for software defect prediction in object-oriented programs[J]. IET Software, 2020.
[375] Geremia S, Tamburri D A. Varying defect prediction approaches during project evolution: A preliminary investigation[C]//2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE). IEEE, 2018: 1-6.
[376] Yang Z, Qian H. Automated Parameter Tuning of Artificial Neural Networks for Software Defect Prediction[C]//Proceedings of the 2nd International Conference on Advances in Image Processing. 2018: 203-209.
[377] Lu H. Semi-supervised and Active Learning Models for Software Fault Prediction[J]. 2015.
[378] Elahi E, Kanwal S, Asif A N. A new Ensemble approach for Software Fault Prediction[C]//2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE, 2020: 407-412.
[379] Cui C, Liu B, Wang S. Isolation Forest Filter to Simplify Training Data for Cross-Project Defect Prediction[C]//2019 Prognostics and System Health Management Conference (PHM-Qingdao). IEEE, 2019: 1-6.
[380] Rosli M. A Framework for Understanding and Evaluating the Quality of Data Sets in Empirical Software Engineering[D]. ResearchSpace@ Auckland, 2018.
[381] Kumar L, Sureka A. Analyzing fault prediction usefulness from cost perspective using source code metrics[C]//2017 Tenth International Conference on Contemporary Computing (IC3). IEEE, 2017: 1-7.
[382] Choeikiwong T, Vateekul P. Two Stage Model to Detect and Rank Software Defects on Imbalanced and Scarcity Data Sets[J]. IAENG International Journal of Computer Science, 2016, 43(3).
[383] Xu Z, Li L, Yan M, et al. A comprehensive comparative study of clustering-based unsupervised defect prediction models[J]. Journal of Systems and Software, 2021, 172: 110862.
[384] Sun Z, Li J, Sun H, et al. CFPS: Collaborative filtering based source projects selection for cross-project defect prediction[J]. Applied Soft Computing, 2021, 99: 106940.
[385] Esteves G, Figueiredo E, Veloso A, et al. Understanding machine learning software defect predictions[J]. Automated Software Engineering, 2020, 27(3): 369-392.
[386] Shatnawi R. Comparison of threshold identification techniques for object-oriented software metrics[J]. IET Software, 2020, 14(6): 727-738.
[387] Mohamed F A, Salama C R, Yousef A H, et al. A Universal Model for Defective Classes Prediction Using Different Object-Oriented Metrics Suites[C]//2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES). IEEE, 2020: 65-70.
[388] Jin C. Cross-project software defect prediction based on domain adaptation learning and optimization[J]. Expert Systems with Applications, 2021, 171: 114637.
[389] Ferenc R, Tóth Z, Ladányi G, et al. A public unified bug dataset for java and its assessment regarding metrics and bug prediction[J]. Software Quality Journal, 2020: 1-60.
[390] Chen X, Mu Y, Liu K, et al. Revisiting heterogeneous defect prediction methods: How far are we?[J]. Information and Software Technology, 2021, 130: 106441.
[391] Pecorelli F, Di Nucci D. Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performances and a benchmark study[J]. Science of Computer Programming, 2021, 205: 102611.
[392] Eken B, Palma F, Ayşe B, et al. An empirical study on the effect of community smells on bug prediction[J]. Software Quality Journal, 2021, 29(1): 159-194.
[393] Alkhaeir T, Walter B. The effect of code smells on the relationship between design patterns and defects[J]. IEEE Access, 2020.
[394] Hassouneh Y, Turabieh H, Thaher T, et al. Boosted Whale Optimization Algorithm With Natural Selection Operators for Software Fault Prediction[J]. IEEE Access, 2021, 9: 14239-14258.
[395] Eivazpour Z, Keyvanpour M R. CSSG: A cost-sensitive stacked generalization approach for software defect prediction[J]. Software Testing, Verification and Reliability, 2021: e1761.
[396] Niu L, Wan J, Wang H, et al. Cost-sensitive Dictionary Learning for Software Defect Prediction[J]. Neural Processing Letters, 2020, 52(3): 2415-2449.
[397] Zhou X, Lu L. Defect Prediction via LSTM Based on Sequence and Tree Structure[C]//2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2020: 366-373.
[398] Zou Q, Lu L, Qiu S, et al. Correlation feature and instance weights transfer learning for cross project software defect prediction[J]. IET Software, 2021, 15(1): 55-74.
[399] Wang A, Zhang Y, Yan Y. Heterogeneous Defect Prediction Based on Federated Transfer Learning via Knowledge Distillation[J]. IEEE Access, 2021, 9: 29530-29540.
[400] Malhotra R, Jain J. Predicting Software Defects for Object-Oriented Software Using Search-based Techniques[J]. International Journal of Software Engineering and Knowledge Engineering, 2021, 31(02): 193-215.
[401] Li X, Yang X, Su J, et al. A Multi-Objective Learning Method for Building Sparse Defect Prediction Models[C]//2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2020: 204-211.

[402] Lavazza L, Morasca S. An Empirical Study of Thresholds for Code Measures[C]//2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2020: 346-357.
[403] Jahanshahi H, Cevik M, Başar A. Moving from cross-project defect prediction to heterogeneous defect prediction: a partial replication study[J]. arXiv preprint arXiv:2103.03490, 2021.
[404] Malhotra R, Jain J. Predicting defects in object-oriented software using cost-sensitive classification[C]//IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2021, 1022(1): 012112.
[405] Wu Y, Yao J, Chang S, et al. LIMCR: Less-Informative Majorities Cleaning Rule Based on Naïve Bayes for Imbalance Learning in Software Defect Prediction[J]. Applied Sciences, 2020, 10(23): 8324.
[406] Wang H, Zhuang W, Zhang X. Software Defect Prediction Based on Gated Hierarchical LSTMs[J]. IEEE Transactions on Reliability, 2021.
[407] Yadav H S. Increasing Accuracy of Software Defect Prediction using 1-dimensional CNN with SVM[C]//2020 IEEE International Conference for Innovation in Technology (INOCON). IEEE, 2020: 1-6.
[408] Yan X, Zhang Y, Khan A A. An algorithm acceleration framework for correlation-based feature selection[C]//MATEC Web of Conferences. EDP Sciences, 2021, 336: 07011.

The list of citations list using the JIRA-HA-2019 / JIRA-RA-2019 data set

[409] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, K. Matsumoto. Predicting Defective Lines Using a Model-Agnostic Technique. arXiv preprint arXiv:2009.03612v1, 2020.
[410] Kushani Perera, Jeffrey Chan, Shanika Karunasekera: A Framework for Feature Selection to Exploit Feature Group Structures. PAKDD (1) 2020: 792-804.
[411] Rajapaksha D, Tantithamthavorn C, Jiarpakdee J, et al. SQAPlanner: Generating Data-InformedSoftware Quality Improvement Plans[J]. arXiv preprint arXiv:2102.09687, 2021.
[412] Jiarpakdee J, Tantithamthavorn C, Grundy J. Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models[J]. arXiv preprint arXiv:2102.12007, 2021.
[413] Jahanshahi H, Cevik M, Başar A. Moving from cross-project defect prediction to heterogeneous defect prediction: a partial replication study[J]. arXiv preprint arXiv:2103.03490, 2021.
[414] Khan S S, Niloy N T, Azmain M A, et al. Impact of Label Noise and Efficacy of Noise Filters in Software Defect Prediction[J].

The list of citations list using the IND-JLMIV+R-2020 data set

[415] Alexander Trautsch, Fabian Trautsch, Steffen Herbold, Benjamin Ledel, Jens Grabowski. The SmartSHARK Ecosystem for Software Repository Mining. arXiv preprint arXiv:2001.01606v1, 2020.
[416] Alexander Trautsch, Steffen Herbold, Jens Grabowski. A Longitudinal Study of Static Analysis Warning Evolution and the Effects of PMD on Software Quality in Apache Open Source Projects. arXiv preprint arXiv:1912.02179v3, 2020.
[417] Steffen Herbold, Alexander Trautsch, Benjamin Ledel. Large-Scale Manual Validation of Bugfixing Changes. MSR 2020.
[418] Herbold, S., (2020). Autorank: A Python package for automated ranking of classifiers. Journal of Open Source Software, 5(48), 2173. https://doi.org/10.21105/joss.02173.
[419] Herbold S, Trautsch A, Trautsch F. On the feasibility of automated prediction of bug and non-bug issues[J]. Empirical Software Engineering, 2020: 1-37.