Received: 31 May 2023 | Revised: 13 January 2024 | Accepted: 30 January 2024

DOI: 10.1002/smr.2715

RESEARCH ARTICLE - EMPIRICAL

Journal of Software: Evolution and Process WILEY

Unveiling the impact of unchanged modules across versions on the evaluation of within-project defect prediction models

Xutong Liu[1] | Yufei Zhou[2] | Zeyu Lu[1] | Yuanqing Mei[1] | Yibiao Yang[1] | Junyan Qian[2] | Yuming Zhou[1]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

[2]School of Computer Science and Engineering and School of Software, Guangxi Normal University, Guilin, China

Correspondence
Yuming Zhou, State Key Laboratory for Novel Software Technology, Nanjing University.
Email: zhouyuming@nju.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Numbers: 62172205, 62162004, U21A20474

Abstract

Background: Software defect prediction (SDP) is a topic actively researched in the software engineering community. Within-project defect prediction (WPDP) involves using labeled modules from previous versions of the same project to train classifiers. Over time, many defect prediction models have been evaluated under the WPDP scenario.

Problem: Data duplication poses a significant challenge in current WPDP evaluation procedures. Unchanged modules, characterized by identical executable source code, are frequently present in both target and source versions during experimentation. However, it is still unclear how and to what extent the presence of unchanged modules affects the performance assessment of WPDP models and the comparison of multiple WPDP models.

Method: In this paper, we provide a method to detect and remove unchanged modules from defect datasets and unveil the impact of data duplication in WPDP on model evaluation.

Results: The experiments conducted on 481 target versions from 62 projects provide evidence that data duplication significantly affects the reported performance values of individual learners in WPDP. However, when ranking multiple WPDP models based on prediction performance, the impact of removing unchanged instances is not substantial. Nevertheless, it is important to note that removing unchanged instances does have a slight influence on the selection of models with better generalization.

Conclusion: We recommend that future WPDP studies take into consideration the removal of unchanged modules from target versions when evaluating the performance of their models. This practice will enhance the reliability and validity of the results obtained in WPDP research, leading to improved understanding and advancements in defect prediction models.

KEYWORDS
data duplication, model evaluation, within-project defect prediction

1 | INTRODUCTION

Software defect prediction (SDP) aims to predict whether modules in software, such as classes, files, and functions, are defective (buggy) or clean. Specifically, code features, including static code metrics,[1] CodeBERT,[2] and process metrics,[3] are extracted from software modules and their corresponding software development processes. These features serve as independent variables, while the binary label indicating whether a module is buggy or clean is considered the dependent variable. The goal of SDP is to assist developers in reducing the effort required for code testing and inspection by assigning different priorities to target modules. The prevalent SDP models are supervised, meaning they rely on labeled training data to construct the predictor.[4] Various sources of training data for SDP models have been explored, including within-project defect prediction (WPDP)[5–7] and cross-project defect prediction (CPDP).[4,8–11] WPDP utilizes labeled modules from previous versions of the same project to train classifiers, while CPDP employs labeled modules from external projects for training. It is generally believed that if there is a sufficient amount of labeled historical modules to train classifiers, WPDP is more suitable for predicting the defectiveness of software modules compared to CPDP.[12] In conclusion, when an ample supply of historically labeled data exists within a project, it is common practice to construct WPDP models to better capture the underlying relationship between metrics and module defectiveness in the target version. Additionally, developers find it more intuitive to gather labeled modules from previous version(s) of the same project to train a classifier and utilize it for predicting new modules in subsequent versions.

WPDP has gained significant attention in the field of SDP research due to its effectiveness and practical value. Previous studies have made notable contributions by introducing novel WPDP models, improving defect prediction performance, and enhancing our understanding of the defect prediction problem.[13] The effects of various factors on model evaluation have been investigated. Several works focus on the data quality of defect datasets and propose new or corrected datasets.[14] The selection of suitable performance indicators is discussed in detail.[15] Several baseline prediction models are proposed to provide a floor performance during model evaluation.[4] These endeavors are furnishing our community with deeper insights into the evaluation of WPDP models.

However, current WPDP studies often suffer from a common issue, namely serious data duplication in the WPDP scenario, which is being omitted by the current WPDP model evaluation methodology. In other words, there are many unchanged modules between the source and target versions (i.e., prior and new versions within a project), which is understandable as numerous modules tend to remain unchanged in software development when transitioning to a new version. Here, "unchanged" means that the executable source code of two modules is identical. In the WPDP context, unchanged modules can potentially lead to two problems. Firstly, in the model evaluation phase, unchanged instances with consistent labels between the source and target versions can introduce overly optimistic estimates of the classifier's performance. Secondly, unchanged instances with inconsistent labels between the source and target versions are likely to introduce noise, which can further degrade the classifier's performance. As a result, the prediction performance of defect prediction models under WPDP may be biased compared to their true performance in the real world. Additionally, the comparison results among multiple prediction models may be biased when compared to the actual situation in the real world.

To date, the impact of unchanged modules on WPDP has not been thoroughly examined. Therefore, in this paper, we first propose a methodology to identify and eliminate unchanged modules from target versions in defect datasets in WPDP model evaluation. We then conduct experiments to investigate the influence of data duplication on the evaluation of WPDP models. Our study is guided by three research questions (RQs). The first RQ focuses on determining the prevalence of data duplication in cross-version defect datasets. In Section 5.1, our findings reveal that, on average, 43.2% of instances in the current version are unchanged compared with their corresponding prior version. This high occurrence of data duplication indicates that it is a common occurrence in current cross-version defect datasets. Given the extensive usage of these datasets in the field of WPDP, the impact of data duplication on WPDP model evaluation is likely to have wide-ranging implications. The second RQ investigates how data duplication affects the prediction performance of WPDP models. The results of our experiments, presented in Section 5.2, demonstrate that data duplication introduces significant bias in the performance evaluation of defect prediction models. Specifically, we observe changes in recall ranging from -19.0% to 22.7%, precision ranging from -14.7% to 25.6%, F1 ranging from -18.3% to 19.6%, G1 ranging from -16.4% to 11.7%, and AUC ranging from -13.9% to 88.6%. The third RQ explores how data duplication impacts the comparison of multiple WPDP models. In Section 5.3, our experimental results indicate that, in general, data duplication during WPDP evaluation does not facilitate the identification of models with good generalization capabilities on unseen modules in target versions. Furthermore, we delve into two additional discussions. First, we explore the underlying reasons why data duplication in WPDP affects the performance evaluation of models. Second, we investigate the possibility of enhancing the generalization of WPDP models to unseen modules in target versions through the utilization of resampling strategies.

The remaining sections of this study are structured as follows: in Section 2, we present an introduction to WPDP and data duplication. Section 3 outlines the methodology used for eliminating unchanged data from cross-version defect datasets. Section 4 describes the experimental setup, while Section 5 reports the results of the experiments. Section 6 is dedicated to discussing our work, and in Section 7, we introduce the related studies. Threats to the validity of our work are analyzed in Section 8. Finally, Section 9 concludes this paper and outlines the direction for future work.

2 | BACKGROUND

In this section, we introduce the core concepts of WPDP and data duplication within WPDP.

2.1 | Within-project defect prediction

In the context of within-project defect prediction (WPDP), the objective is to predict the defectiveness of modules using prediction models trained on historical modules within the same project. One common approach within WPDP is cross-version defect prediction (CVDP), which involves utilizing historical data from previous project versions to construct predictive models capable of identifying potential defects in the current version. CVDP can be further classified into various scenarios based on the selection of training data. Amasaki[16] identified three CVDP scenarios: SFV (single farthest version), SPV (single prior version), and APV (all prior versions). SFV involves using project data from the farthest (oldest) accessible prior version to construct defect prediction models. SPV utilizes project data from the latest accessible prior version for model construction. APV utilizes project data from all accessible prior versions. The specific focus of our research is to investigate the presence of unchanged modules in consecutive project versions and its impact on the performance evaluation of WPDP models. Therefore, our experimental study concentrates on the SPV CVDP scenario, where we examine two consecutive versions from our defect datasets.

In WPDP studies,[16–21] traditional machine learning classifiers are commonly employed. These classifiers include k-nearest neighbors (KNN),[22] logistic regression (LR),[23] Naive Bayes (NB),[24] random forest (RF),[25] and support vector machine (SVM).[26] In our research, we utilize all five of these classifiers to investigate the impact of data duplication on the performance evaluation in WPDP. It is worth noting that we opt for these basic classifiers rather than more complex state-of-the-art classifiers used in recent studies. The reason behind this choice is that simpler classifiers are easier to comprehend and interpret compared to their complex counterparts. Compared to deep learning-based defect prediction models, traditional machine learning classifiers typically have fewer parameters and straightforward algorithms, which facilitates the identification and analysis of the impact of data duplication on performance metrics. The transparency provided by these simple classifiers aids in gaining insights into the underlying mechanisms and potential issues associated with data duplication. This advantage of using simpler classifiers can contribute to a deeper understanding of how data duplication affects classification performance.

2.2 | Data duplication in WPDP

In software engineering research, the duplication of data across sets—where instances appear in both training and test sets—has been recognized as a significant factor that distorts experimental settings and inflates performance metrics.[27,28] For instance, Allamanis, in,[28] delved into the adverse impacts of code duplication on machine learning models across various tasks like code completion, type prediction, and code summarization. The findings revealed that "reported performance metrics are sometimes artificially boosted by up to 100% when testing on duplicated code corpora compared to de-duplicated corpora, which more accurately simulate how machine learning models operate on code used by software engineers." Similarly, in,[27] Kang et al observed the prevalence of warnings present in both training and testing datasets when assessing cross-version models for predicting false static warnings. They highlighted that this scenario contributes to an unrealistic experimental setup and emphasized the need to deduplicate test data from the training dataset to gauge the genuine effectiveness of prediction models.

Our study centers on defect prediction and resonates with prior research in this domain. Allamanis, in,[28] emphasized a pivotal objective within software engineering: the proficient utilization of trained models to offer insights and recommendations for new, unseen code during its creation or maintenance. This imperative necessitates the robust adaptability of machine learning models to navigate through novel instances of source code effectively. The presence of "cross-set" data duplication poses a substantial threat to a model's performance and reliability. When identical data exists in both the training and test sets, it can artificially inflate performance metrics, creating a distorted impression of the model's actual capabilities. Notably, "cross-set" data duplication fundamentally contradicts this principle.

In the context of WPDP, data duplication occurs when the same instances appear in both the source and target versions. Specifically, data duplication refers to unchanged modules that exhibit identical executable source code in both the source and target versions (specifically, consecutive versions in the SPV scenario). Note that, in this study, "unchanged" does not refer to modules with identical features at the feature level but rather identical modules at the source code level. Because different source code modules may occasionally exhibit identical values on code metrics. For instance, two distinct functions may possess the same lines of code (LOC).

Assuming module B in the test data as an unchanged version of module A in the training data. This creates two cases when comparing their labels:

1. Consistent labels: When A and B have matching labels, including B in the test data can inflate the prediction performance of the trained model. This happens because the relationship between the instance and label has already been seen in the training data.

2. Inconsistent labels: If A and B have conflicting labels, our prior research identified mislabeling and/or extrinsic bugs as primary causes.[29] We have categorized these scenarios:

- No Bug Scenario: inconsistent labels indicate a mislabeling issue, suggesting partial correctness in these labels.
- Intrinsic Bug Scenario: inconsistent labels signify mislabeling with partial correctness.
- Extrinsic Bug Scenario: inconsistencies do not necessarily imply mislabeling but could indicate external factors.[30]

For inconsistent labels in a multi-version project dataset, three possible cases emerge: (1) all-incorrect: all involved labels are incorrect; (2) partly-correct: the involved labels comprise both correct and incorrect labels; and (3) all-correct: all involved labels are correct. In the following, we examine what inconsistent labels mean for defect prediction.

- Both labels are incorrect. A represents a mislabeled instance in the training set, leading to bias if used to build a defect prediction model. Similarly, B, as a mislabeled instance in the test data, distorts the evaluation of the model's performance. Here, A represents noise in the training set, while B signifies noise in the test set.
- Only one label is correct. If B holds an incorrect label, it becomes noise in the test set. From the training set's perspective, A might not be considered noise as it has a correct label. Conversely, if A has an incorrect label, it's noise in the training data, while B might not be considered noise from the test set's viewpoint.
- Both labels are correct. This scenario arises due to an extrinsic bug. Traditional defect prediction models aim to explore the predictability of intrinsic bugs using features like code and process metrics. They do not include external factors causing extrinsic bugs. Hence, defects caused by external factors should not be used to study intrinsic bug associations. Instances with inconsistent labels due to an extrinsic bug should be treated as noise, regardless of their set. This aligns with a recent study by Rodríguez-Pérez et al,[2] advocating for the removal of extrinsic bugs when evaluating defect prediction models.

Hence, encountering instances with inconsistent labels suggests at least one instance is noise for defect prediction. Determining which instance(s) constitutes noise is challenging without additional information. In practical terms, a pair of instances with inconsistent labels presents a conflicting relationship between instance features and class labels. Thus, from a practical standpoint, both instances with inconsistent labels can be considered "noise" data.[29] The prevalence of unchanged modules with inconsistent labels in widely used defect datasets, such as ECLIPSE-2007, JIRA-RA-2019, and Metrics-Repo-2010 (details in Section 6.1), may underscore the persistent and extensive influence of resulting bias in performance evaluation on a range of future studies relying on these widely used datasets.

Considering the aforementioned reasons, we advocate for removing unchanged instances from the test data to achieve a more accurate assessment of the defect prediction model's performance during evaluation. Importantly, this recommendation does not intend to confine defect prediction solely to newly added or modified modules during the model application stage.

3 | METHODOLOGY OF EVALUATING WPDP MODELS WITHOUT DATA DUPLICATION

In this section, we present our approach for identifying and eliminating unchanged modules in order to assess SDP modules devoid of any duplicated data. Initially, we present a practical instance of code comparison, elucidating the standards employed to ascertain if the two modules are identical. Subsequently, we outline the procedure for removing unchanged modules from the target version in WPDP.

Figure 1 illustrates our approach for detecting unchanged modules based on specific criteria. In the example provided, we examine the ITestRunView.java file, which exists in both Eclipse-2.0 (on the left) and Eclipse-2.1 (on the right). Our objective is to identify modules where the source code remains unchanged in the new version. To achieve this, we disregard text differences in code comments and focus solely on non-blank discrepancies in executable source code between the compared modules. In Figure 1, the blue-highlighted text differences are ignored as they represent code comments. The red-highlighted lexemes differences play a pivotal role in determining whether the two code segments are distinct. We observe that in Eclipse-2.1, there are differences in lexemes (specifically, lines 23, 48, and 53) compared to its counterpart in Eclipse-2.0. Consequently, the ITestRunView.java file in Eclipse-2.1 is deemed not to be an unchanged module when compared to its previous version in Eclipse-2.0.

In the following, we describe the process of identifying all unchanged modules within a target version in relation concerning the source version in WPDP. Figure 2 describes this overflow:

1. To compare modules between a source version (v1) and a target version (v2) in a project, we automatically extract files, classes, functions, and their respective lexemes from each version. Using Scitool Understand, we create an Understand Database (udb) dataset for each version. The udb dataset provides crucial information for identifying unchanged modules and conducting detailed comparisons between the two versions.

Text Compare

File: plugins\org.eclipse.jdt.junit\src\org\eclipse\jdt\internal\junit\ui\ITestRunView.java

1	/*	<>	1	/**********… » **********
2	* (c) Copyright IBM Corp. 2000, 2001.		2	* Copyright (c) 2000, 2003 IBM Corpor » ation and others.
3	* All Rights Reserved.		3	* All rights reserved. This program a » nd the accompanying materials
			4	* are made available under the terms » of the Common Public License v1.0
			5	* which accompanies this distribution » , and is available at
			6	* http://www.eclipse.org/legal/cpl-v1 » 0.html
4	*/		7	*
			8	* Contributors:
			9	* IBM Corporation - initial API a » nd implementation
			10	**********… » **********
				» ******/
17	public String getTestName();	<>	23	public String getSelectedTestId();
42	public void setSelectedTest(String » testName);	<>	48	public void setSelectedTest(String » testId);
47	public void endTest(String testNam » e);	<>	53	public void endTest(String testId) » ;

FIGURE 1 Code comparison between two files that have the same names in Eclipse-2.0 vs. Eclipse-2.1.

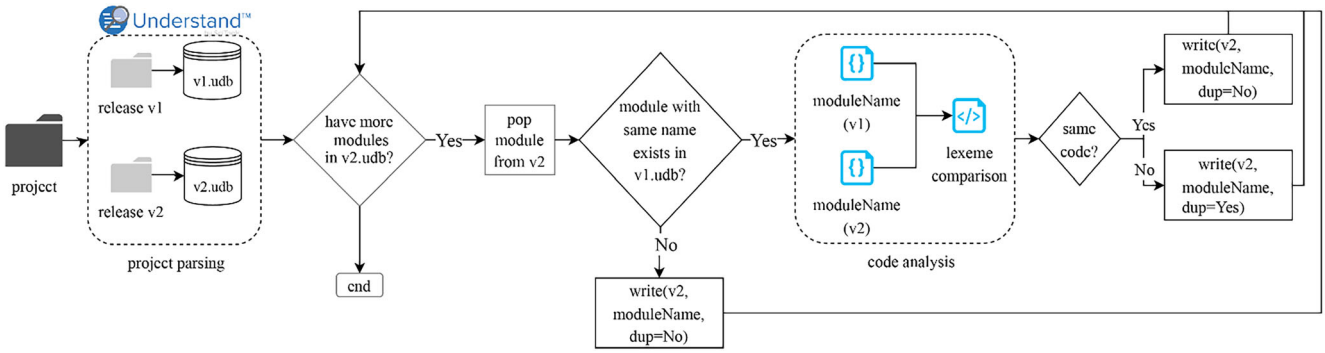

FIGURE 2 The workflow of identifying and removing unchanged modules in a target version.

2. Once the parsing is complete, we proceed by retrieving one module from the udb dataset of the target version (v2). If there are still unprocessed modules remaining in v2, we continue this iteration. However, if all modules in v2 have been processed, the process ends.
3. Upon retrieving a module from the target version (v2), we search for a module with the same name in the source version (v1). If a matching module is found, we proceed to step (4). However, if no module with the same name is found, the popped module is deemed not an "unchanged module" in the target version. In such cases, we record this information and write it into a file. We then return to step (2) to continue the process with the next module in v2.
4. Once two modules with the same name are identified in both v1 and v2, a code analysis is performed to compare their non-blank executable source code. If the code is found to be identical, the module is considered an "unchanged module" in the target version. Conversely, if the code differs, the module is determined not to be an "unchanged module" in the target version. In either case, the corresponding information is recorded and written into a file. Afterward, we proceed to step (2) to continue the process with the next module in v2.

By following the aforementioned process, it becomes possible to identify all unchanged modules within the target versions in WPDP. Subsequently, when evaluating a prediction model trained on source version v1, it is advantageous to remove these unchanged modules from the

corresponding target version v2. This step ensures a non-biased performance evaluation for the model on v2. To demonstrate the effectiveness of our proposed workflow, we record the time spent on identifying unchanged modules in a target version. Figure 2 represents the distribution of time spent across all target versions in each dataset used in our experiment (detailed information regarding the datasets can be found in Section 4.2). From Figure 2, we can find that except for ECLIPSE-2007, the average time cost for each target project on the other three datasets is less than 25 s. Since the target versions in ECLIPSE-2007 contain a large number of files, the process of identifying unchanged modules requires more time, 60s on average; however, this remains within an acceptable range. Overall, our workflow for identifying unchanged modules validates to be efficient.

In Figure 1, we use a Java file across two versions of a project as a motivation example of WPDP modules with data duplication. However, the granularity of data duplication corresponds to the granularity of the defect dataset, which means that if the granularity of the defect dataset is "function", then the data duplication in WPDP will indicate unchanged function across consecutive releases in a project. That's why we use the term "module" to indicate that our methodology can be generalized to different granularity of software modules. Specifically, we conduct our methodology on Java files (datasets ECLIPSE-2007, JIRA-RA-2019, IND-JLMIV+R-2020, and MA-SZZ-2020) and Java classes (dataset Metrics-Repo-2010) in our experiments. Nevertheless, please note that in this paper, defect prediction specifically refers to predicting defects at the source code granularity, such as in Java source code, rather than on other code representation granularity such as byte code. The main reason is that defect prediction at the source code granularity supports developers in directly modifying their suspicious code based on the predictive results. As a result, it is the most extensively studied area in the field of defect prediction research. Therefore, we align with current mainstream defect prediction research by investigating the effectiveness of defect prediction models at the source code granularity. Specifically, our research focuses on assessing the impact of unchanged modules on the evaluation effectiveness of source code granularity WPDP models.

In Sections 5.2 and 5.3, we will utilize the described process to examine the influence of data duplication on model performance evaluation and model ranking. This investigation will provide insights into the impact of unchanged modules on the accuracy and effectiveness of the prediction models used in WPDP.

4 | EXPERIMENTAL SETUP

In this section, we describe the experimental setup for evaluating the impact of unchanged modules on WPDP performance, including the research questions, datasets utilized, performance indicators employed, and the statistical tests conducted.

4.1 | Research questions

In our study, we have formulated the following research questions (RQs):

- RQ1: How prevalent is data duplication in cross-version defect datasets?

The objective of RQ1 is to assess the prevalence of data duplication within existing cross-version defect datasets that have been utilized by researchers. Our analysis aims to determine the extent to which unchanged data instances are present in these datasets, shedding light on the potential impact of data duplication on research outcomes in the field of cross-version defect analysis.

- RQ2: How does data duplication influence the prediction performance of WPDP models?

The objective of RQ2 is to assess the impact of data duplication on the prediction performance of WPDP models. Specifically, we compare the performance of prediction models on the same target version, considering both scenarios with and without unchanged modules. By conducting this comparison, our goal is to investigate whether data duplication results in an optimistic estimation or pessimistic estimation of the model's performance during evaluation. Note that in this paper, we use "optimistic estimation" and "pessimistic estimation" to refer to two distinct scenarios: duplication-induced optimistic estimation of model performance and duplication-induced pessimistic estimation of model performance. More precisely, the former pertains to cases where a sample previously seen in the training set reappears in the test set with the same label. This can lead to the model being evaluated on samples it has already encountered in the training set, potentially resulting in an optimistic estimation of its predictive performance, especially if there are many such "unchanged instances with consistent labels" The latter scenario involves a sample seen in the training set reappearing in the test set with the opposite label. This can also lead to the model being evaluated on samples it has already encountered in the training set, potentially resulting in a pessimistic estimation of its predictive performance, especially if there are many such instances of "unchanged instances with inconsistent labels." Analysis of RQ2 will provide valuable insights into the effects of data duplication on the accuracy and reliability of WPDP models, ultimately enhancing our understanding of the potential biases introduced by unchanged modules in the evaluation process.

- RQ3: How does data duplication influence the comparison results between multiple WPDP models?

The objective of RQ3 is to examine how data duplication affects the comparison results between multiple WPDP models. We aim to investigate whether the ranking of models in terms of prediction performance can be biased by the presence of unchanged modules. Specifically, we compare the performance rankings of WPDP models before and after removing unchanged modules from the target versions. By conducting this analysis, we seek to understand whether data duplication has a significant impact on the relative performance rankings of WPDP models. This investigation will provide insights into the potential biases introduced by unchanged modules, allowing for a more accurate and unbiased comparison of different WPDP models.

4.2 | Prediction models

In our study, we investigate RQ1 to RQ3 using various defect prediction approaches, including k-nearest neighbors (KNN), logistic regression (LR), Naive Bayes (NB), random forest (RF), and support vector machine (SVM). We directly utilize the implementations of these models provided by the scikit-learn Python library, without altering any default parameters, which is a common setting in current defect prediction studies.[31] We choose the above traditional machine learning models for the following reasons: first, they are the most commonly used prediction classifiers in software defect prediction studies.[13,32] Second, compared to more complex deep learning-based classifiers used in recent studies,[33] traditional machine learning classifiers are typically easy to explain and have a relatively stable performance without a large effort for tuning, which facilitates the identification and analysis of the impact of data duplication on performance metrics. In contrast, deep learning models, especially deep neural networks, can have a much larger number of hyperparameters due to the intricate architecture and numerous layers involved. These hyperparameters might include the number of layers and the number of neurons in each layer, the choice of activation functions. In practice, the higher number of hyperparameters in deep learning models often leads to a more complex and time-consuming tuning process compared to traditional machine learning models.

Additionally, we incorporate an automated machine learning (AutoML) tool called autogluon (Auto),[34] which utilizes state-of-the-art deep learning techniques and other machine learning algorithms to train high-performance models. One significant factor driving the choice of autogluon lies in its unique capability to autonomously compose high-performing models from an extensive array of available deep-learning and machine-learning models. That is to say, autogluon may offer a unique blend of deep learning's predictive potency and the practicality and efficiency of traditional machine learning. Therefore, besides traditional machine learning classifiers, we include autogluon in our experiments as a prediction classifier. To better suit the defect prediction task, we customize the settings of AutoML. Specifically, we set the goal metric during optimization as "F1" instead of the default "accuracy." Furthermore, we adjust the optimization level to "best quality" from the default "medium quality." The rest hyperparameters are default values of autogluon.

Data normalization is applied prior to applying any prediction approach. For a comprehensive overview of the configurations used in our work, please refer to Table 1. The first column represents the prediction step, while the second and last columns indicate the options and specific classes within the public Python libraries employed in our study.

4.3 | Datasets

In our experiment, we utilize five cross-version datasets: ECLIPSE-2007,[35] Metrics-Repo-2010,[36–38] JIRA-RA-2019,[39] IND-JLMIV+R-2020,[40] and MA-SZZ-2020.[41] Table 2 provides a summary of the statistical information for these datasets. The first and second columns of Table 2 indicate the dataset and project names, respectively. The third column presents the versions available in each project. The fourth and fifth columns display the range of the number of instances and the range of the ratio of defective modules within each dataset before eliminating unchanged modules.

TABLE 1 Configurations.

Step	Option	Algorithm
Data normalization	Standardization	sklearn.preprocessing. StandardScaler
Classification	Logisic Regression	sklearn.linear_model.LogisticRegression
	Support Vector Machine	sklearn.svm. SVC
	Naïve Bayes	sklearn.naive_bayes.GaussianNB
	Ramdom Forest	sklearn.ensemble. RandomForestClassifier
	KNN	sklearn.neighbors.KNeighborsClassifier
	Autogluon	autogluon. TabularDataset

TABLE 2 Dataset information.

Dataset	Project	Versions	#instances	%defective	#metrics	Type
ECLIPSE-2007	eclipse	2, 2.1, 3	6,729 ~ 10,593	11% ~ %15	198	code metrics
IND-JLMIV+R-2020	38 projects	395 versions	3-1708	0% ~ 36%	44	code metrics
JIRA-RA-2019	activemq	5, 5.1, 5.2, 5.3, 5.8	1884 ~ 3,420	6% ~ %16	65	code, process, own. Metrics
	camel	1.4, 2.1, 2.11, 2.9	1,515 ~ 8,846	2% ~ %19		
	derby	10.2, 10.3, 10.5	1963 ~ 2,705	14% ~ %34		
	groovy	1.5, 1.6, 1.6	757 ~ 884	3% ~ %9		
	hbase	0.94, 0.95, 0.95	1,059 ~ 1834	21% ~ %26		
	hive	0.1, 0.12, 0.9	1,416 ~ 2,662	8% ~ %20		
	jruby	1.1, 1.4, 1.5, 1.7	731 ~ 1,614	5% ~ %18		
	lucene	2.3, 2.9, 3, 3.1	805 ~ 1831	6% ~ %24		
	wicket	1.3, 1.3, 1.5	1,672 ~ 2,578	4% ~ %7		
MA-SZZ-2020	flume	1.2, 1.3, 1.3, 1.4, 1.5, 1.5, 1.5, 1.6, 1.7, 1.8	272 ~ 574	4% ~ %32	44	code metrics
	mahout	0.1, 0.11, 0.12, 0.13, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	1,027 ~ 1,267	17% ~ %29		
	maven	2.2, 2.2, 3, 3, 3, 3, 3, 3, 3.1, 3.1	318 ~ 703	4% ~ %13		
	shiro	1.1, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.3, 1.3	381 ~ 493	3% ~ %7		
	zeppelin	0.5, 0.5, 0.5, 0.6, 0.6, 0.6, 0.7, 0.7, 0.7, 0.7	129 ~ 413	21% ~ %38		
Metrics-Repo-2010	ant	1.3, 1.4, 1.5, 1.6, 1.7	125 ~ 745	11% ~ %26	20	code metrics
	camel	1, 1.2, 1.4, 1.6	339 ~ 965	4% ~ %36		
	forrest	0.6, 0.7, 0.8	6 ~ 32	6% ~ %17		
	jedit	3.2, 4, 4.1, 4.2, 4.3	272 ~ 492	2% ~ %33		
	log4j	1, 1.1, 1.2	109 ~ 205	25% ~ %92		
	lucene	2, 2.2, 2.4	195 ~ 340	47% ~ %60		
	pbeans	1, 2	26 ~ 51	20% ~ %77		
	poi	1.5, 2, 2.5, 3	237 ~ 442	12% ~ %64		
	synapse	1, 1.1, 1.2	157 ~ 256	10% ~ %34		
	velocity	1.4, 1.5, 1.6	196 ~ 229	34% ~ %75		
	xalan	2.4, 2.5, 2.6, 2.7	723 ~ 909	15% ~ %99		
	xerces	0, 1.2, 1.3, 1.4	162 ~ 588	15% ~ %74		

The sixth column specifies the number of metrics included in each dataset. The last column describes the metric types included in each dataset. Among these datasets, ECLIPSE-2007, Metrics-Repo-2010, and JIRA-RA-2019 are commonly utilized datasets in the field of defect prediction. MA-SZZ-2020 and IND-JLMIV+R are two newer defect datasets published in 2020. It is worth noting that the original IND-JLMIV+R-2020[40] dataset contains 4,198 metrics. However, considering the potential issue of overfitting due to the high number of metrics compared to the number of instances, we have reduced it to 44 process metrics for our experiments (for more details, please refer to our replication kit[42]). From Table 2, it is evident that the number of instances and the ratio of defective modules vary significantly across different versions within each dataset, providing a diverse range of scenarios for our experiments.

4.4 | Model evaluation

In our investigation of the impact of data duplication on WPDP model performance evaluation, we utilize several performance indicators to assess the models. These indicators include recall, F_1 score, G1 score, and AUC (area under the curve). To calculate these performance indicators, we

utilize a confusion matrix based on the predictions for instances in a target version. The components of the confusion matrix are as follows: TP (true positive, the number of instances correctly predicted as positive), FP (false positive, the number of instances wrongly predicted as positive), TN (true negative, the number of instances correctly predicted as negative, and FN (false negative, the number of instances wrongly predicted as negative). Using the values from this confusion matrix, we use performance indicators including Recall, Precision, F1 score, G1 score, and AUC in our experiments.

- Recall. It is a performance indicator that measures the ratio of positive instances (i.e., buggy modules) that are correctly predicted as positive. It is defined as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1}$$

By calculating recall, we can assess the ability of a prediction model to identify positive instances correctly, highlighting its effectiveness in capturing true positive cases.

- F_1. It is the harmonic mean of precision and recall. It is defined as:

$$\text{F}_1 = 2 * \text{precision} * \frac{\text{recall}}{\text{precision} + \text{recall}} \tag{2}$$

where precision is the ratio of the number of correctly predicted positive instances to the total number of instances that are predicted to be positive:

$$precision = \frac{TP}{TP + FP} \tag{3}$$

F_1 takes into account both precision and recall to provide a single measure of a model's performance. Although researchers argued that F_1 cannot reflect TN[43] and is hard to explain, it is still one of the most frequently used performance indicators in SDP. Actually, according to a systematic review, 53% of SDP literature used F_1 over the period 2012–2022.[15]

- G_1. It is a performance indicator commonly used in binary classification problems. This indicator is formulated based on taking into account both recall and false alarm, thereby showing how much balance between the two indicators is achieved overall.[15] It is defined as:

$$G_1 = \sqrt{recall * (1 - FPR)} \tag{3}$$

where FPR is:

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{4}$$

G_1 is an imbalance-independent performance indicator, which is independent of the proportion of positive (and negative) elements in the dataset,[44] and therefore being recommended by previous defect prediction studies.[45]

- AUC (area under the curve). It measures the area under the receiver operating characteristic (ROC) curve, which plots the true positive rate (recall) against the false positive rate (FPR) for different classification threshold values. The ROC curve provides insights into the trade-off between the true positive rate and the false positive rate at various thresholds. AUC summarizes the overall performance of the classifier by quantifying the area under this curve. The AUC value ranges between 0 and 1, where a value of 0.5 indicates a classifier that performs no

better than random guessing, while a value of 1 indicates a perfect classifier. Higher AUC values indicate better performance of the classifier in distinguishing between the positive and negative classes.

Our decision is grounded in a comprehensive evaluation strategy. In a systematic review of 111 publications (2012-2022),[15] AUC (54%), F1 (53%), recall (51%), precision (39%), FPR (17%), and G1 (11%) were identified as widely used in defect prediction. Furthermore, this review indicated that 57% of the analyzed publications utilized at least three indicators, reflecting the common practice of employing multiple metrics for a more thorough assessment. Additionally, within the specific context of software defect prediction (SDP), comparative analysis suggests that a technique excelling in one metric may perform differently in another.[15] To account for this complexity, we aim to provide a comprehensive evaluation of our experiments based on the above five commonly used performance indicators and avoid drawing conclusions based solely on one or two metrics.

In order to quantify the impact of data duplication on the performance evaluation of WPDP models, we calculate the performance difference between the predictions made on the same target project with and without unchanged modules. The definition of such a performance delta is:

$$\text{delta}_{\text{indicator}} = \frac{\text{indicator}_{\text{no-dup}} - \text{indicator}_{\text{origin}}}{\text{indicator}_{\text{origin}}} \times 100\% \tag{5}$$

Here, $indicator_{no\text{-}dup}$ represents the prediction performance on the target version without unchanged modules, which involves only predicting unseen modules. $indicator_{orgin}$ represents the prediction performance on the original target version, which includes unchanged modules. If the performance delta ($delta_{indicator}$) is larger than 0, it indicates an optimistic estimation caused by data duplication, suggesting that the presence of unchanged modules inflated the performance evaluation. Conversely, if the performance delta is smaller than 0, it indicates a pessimistic estimation caused by data duplication, implying that the presence of unchanged modules led to a lower performance evaluation than the actual performance. By calculating and analyzing the performance delta, we can better understand the influence of data duplication on the performance evaluation of WPDP models in the evaluation process.

4.5 | Statistical test

In our study, we employ the Wilcoxon signed rank test[46] as a statistical test to determine the significance of performance differences between different experimental settings, such as the performance of a model on the target version with or without unchanged modules. The test helps us assess whether the observed differences are statistically significant. We set the significance level to 0.05, indicating that a p-value below this threshold is considered statistically significant. Additionally, we report the effect size, which provides a measure of the magnitude of the performance difference. The effect size Cohen's d is calculated using a formula defined in[47]: $d = Z/\sqrt{N}$. Here, Z represents the absolute value of the sum of ranks for the differences between the paired observations in the Wilcoxon signed-rank test, and N is the total number of paired samples. The effect size is considered "negligible" for$|d| < 0.1$, "small" for $0.1 \le |d| < 0.3$, "medium" for $0.3 \le |d| < 0.5$, otherwise for "large". By applying the Wilcoxon signed rank test and reporting the effect size, we can assess the statistical significance and practical significance of the performance differences observed in our experiments.

We utilize the Non-Parametric Scott-Knott Effect Size Difference (NPSK) test[48,49] to compare the performances of multiple prediction models and rank them accordingly. The NPSK test is a non-parametric approach that does not rely on assumptions about the underlying data distribution. It has been widely used in previous software defect prediction (SDP) studies.[50–52] The NPSK test allows us to divide the models into distinct groups based on statistically significant differences. It ensures that within each group, the differences in performance indicators among the models are negligible, while the differences between groups are non-negligible. This approach provides a reliable ranking of WPDP models. In Section 5.3 of our study, we apply the NPSK test using a two-level approach. First, for each dataset, we consider all target versions as samples and treat the models as treatments. The performance of each model on each target version is treated as an observation. We perform the NPSK test to calculate the performance ranking of the models for each dataset. This is the one-level NPSK test. Second, we treat the ranking results from each dataset as observations and consider all datasets in our experiment as samples. We perform the NPSK test again to calculate the overall ranking of models across all datasets. This is the two-level NPSK test. By conducting the NPSK test, we obtain the ranking of models based on their performance across the entire dataset in our experiment. Additionally, in Section 5.3, we use the Wilcoxon signed-rank test to examine the statistical significance of the rankings between the settings with and without unchanged modules. By combining the one-level NPSK rankings from all datasets into observations, we can determine if the rankings of multiple models in the compared groups (with and without data duplication) are significantly different. A p-value below 0.05 suggests a significant difference in the ranking results between the two groups.

5 | EXPERIMENTAL RESULTS

In this section, we report the experimental results in detail.

5.1 | RQ1: how prevalent is data duplication in cross-version defect datasets?

Table 3 presents the average duplication ratio across test sets in our study. The first three columns display the dataset name, the total number of projects in this dataset, and the total number of versions in this dataset, respectively. The fourth column represents the average number of instances per test set (i.e., per target version) across all test sets in a dataset. Under the SPV WPDP scenario, each target release is predicted by its previous release (if available). For example, assume there are three releases in a project, release 1, 2, and 3, then there will be 2 pairs of training-testing releases: training on release 1 and testing on release 2; training on release 2 and testing on release 3. Therefore, for a dataset that contains k projects and n releases, then there will be n − k training-testing release pairs. For example, the total number of test sets of MA-SZZ-2020 is 45 (50 releases − 5 projects). Our experiments are conducted on a total of 458 test sets (523 releases − 65 projects). The fifth column represents the average duplication ratio across all test sets in a project, which is calculated as the number of modules that are unchanged compared with their previous version divided by the total number of modules in a target version. The sixth and seventh columns display the average ratio of positive modules to all unchanged modules and the average ratio of negative modules to all unchanged modules across all test sets in a dataset, respectively. The eighth column represents the average class ratio across all tests in a dataset, calculated as the number of negative modules divided by the number of positive modules, per target version with unchanged modules. The last column displays the average class ratio per test set without unchanged modules. The last row of Table 3 reports the corresponding values on all five datasets.

Furthermore, Figure 3 illustrates the distribution of the ratio of unchanged modules to modules across all 458 test sets in five datasets.

Based on Table 3 and Figure 3, we make the following two interesting observations.

- *Observation 1: The overall average duplication ratio across all 458 test sets is 43.2%*. This indicates that a significant proportion of modules in target versions are unchanged compared with their previous versions. The high average duplication ratio suggests that data duplication is prevalent in public cross-version defect datasets used in WPDP. Figure 4 reveals that both positive modules and negative modules have high ratios of unchanged modules. This implies that data duplication affects both defective and non-defective modules in the target versions. The presence of unchanged modules in both positive and negative instances suggests that the impact of data duplication is not limited to a specific class of modules and can influence the performance evaluation of WPDP models for both defective and non-defective modules.
- *Observation 2: a significant majority of unchanged modules are negative (clean) modules, as indicated by the average negative ratio of unchanged modules being 93.5% across all 458 test sets*. This means that the majority of unchanged modules in target versions are non-defective. Additionally, we find that the class ratio decreases after removing unchanged modules. In other words, the ratio of negative modules in unchanged modules is higher compared to the ratio of negative modules in the entire target version. This phenomenon can be understood based on the nature of software development, where clean modules are more likely to remain unchanged in subsequent versions, while buggy modules are more likely to be fixed. Moser et al's study[53] also corroborates our viewpoint. They found that change metrics (such as the number of revisions of a file, the number of times a file has been refactored) outperform process metrics in defect prediction, which can be explained as "if a file remains unchanged (will be identified as unchanged modules in our study) throughout its life cycle, the change metrics may correctly predict it as non-defective". In summary, understanding the label distribution of unchanged modules is crucial for analyzing the impact of data duplication on WPDP model performance evaluation. Further details and analysis regarding this observation will be discussed in Section 6.1.

Conclusion. Data duplication is widespread in public cross-version defect datasets, with a high percentage of unchanged modules in target versions. This may lead to a significant impact on WPDP performance evaluation.

TABLE 3 The average value of statistics across consecutive versions of datasets.

Dataset	#projects	#versions	Ins	Dup%	Pos%	Neg%	avg_classRatio	avg_classRatio_noDup
ECLIPSE-2007	1	3	9240.5	22.5	2.6	97.4	7.0	5.4
IND-JLMIV+R-2020	38	395	393.6	41.2	3.5	96.5	31.3	20.3
JIRA-RA-2019	9	32	2629.3	39.6	3.3	96.7	13.4	8.6
MA-SZZ-2020	5	50	613.4	71.7	13.2	86.8	9.4	4.3
Metrics-Repo-2010	12	43	407.8	31.1	27.7	72.3	4.6	3.7
Total	65	523	665.7	43.2	6.5	93.5	25.5	16.6

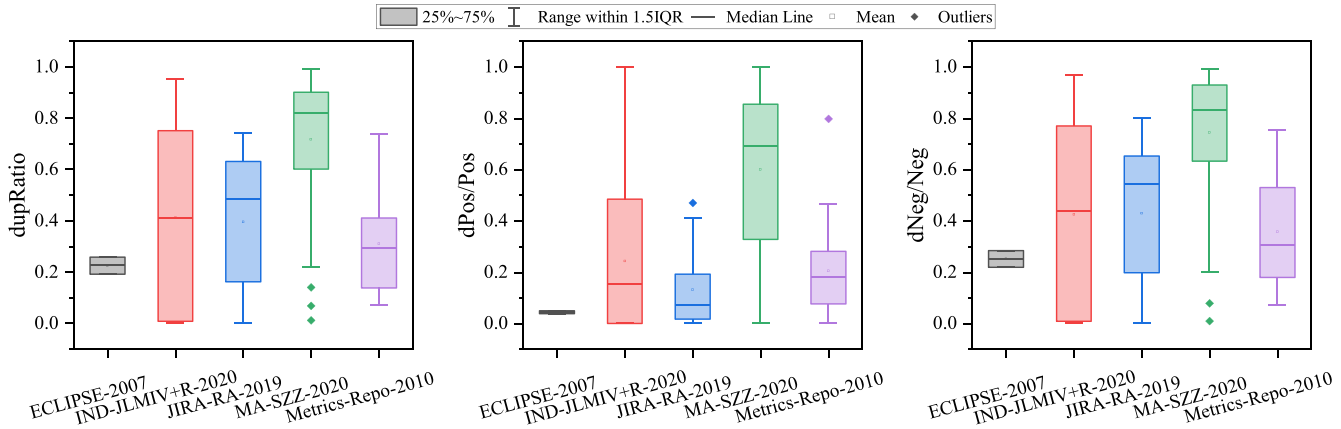

FIGURE 3 Distributions of (1) the ratio of unchanged modules to all modules in a target version, (2) the ratio of unchanged positive modules to all positive modules in a target version, and (3) the ratio of unchanged negative modules to all negative modules in a target version.

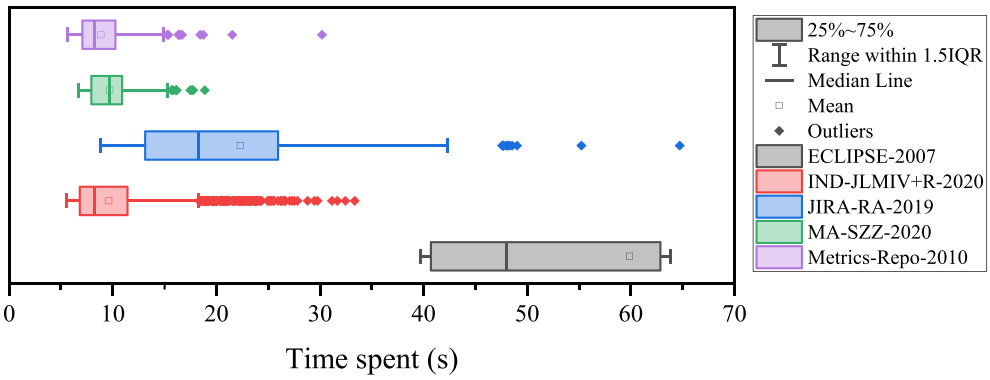

FIGURE 4 The distribution of time spent on identifying unchanged modules in a target version.

5.2 | RQ2: how does data duplication affect the prediction performance of WPDP models?

Table 4 provides the mean performance delta of WPDP models on each dataset, including precision (P), recall (R), F1, G1, and AUC. It also includes the performance delta of TP, FP, TN, and FN, which allows for a more detailed analysis of performance changes when unchanged modules are removed from the target version. Table 5 presents the Benjamini-Hochberg corrected p-values from the Wilcoxon signed rank test and the effect sizes, providing insights into the statistical significance and magnitude of performance differences between the target version with and without unchanged modules.

From Tables 4 and 5, we make the following observations:

- *Observation 1: after removing unchanged modules from target versions, there is a noticeable change in the performance of WPDP models.* Table 4 demonstrates this change, indicating that the recall, precision, F1, G1, and AUC values can vary significantly. The range of change in recall is from -14.7% to 25.6%, in precision from -19.0% to 22.7%, in F1 from -18.3% to 19.6%, in G1 from -16.4% to 11.7%, and in AUC from -13.9% to 88.6%. These findings highlight the impact of data duplication on the performance evaluation of WPDP models.
- *Observation 2: in the majority of cases, there is a statistically significant change in the performance of WPDP, with effect sizes ranging from "medium" to "large".* The performance differences between predictions on the target version with and without unchanged modules are statistically significant in most cases, as indicated by the p-values from the Wilcoxon signed rank test. Additionally, the effect size values demonstrate a non-trivial impact of unchanged modules on the performance evaluation. Taking recall as an example, the p-values of KNN, LR, NB, RF, and SVM for all four datasets are below 0.05, and the effect sizes are generally categorized as "medium" or "large." These results emphasize the significance of considering data duplication in the performance evaluation of WPDP models.
- *Observation 3: in most cases, removing unchanged modules from target versions results in an improvement in the prediction performance of WPDP models.* When considering precision, recall, F1, and G1, most WPDP models demonstrate increased performance after the removal of

TABLE 4 The mean value of performance delta of prediction performance (target version without duplication vs. original target version) in percentage.

Dataset	Model	R	P	F1	G1	AUC	FN/N	FP/N	TN/N	TP/N
ECL	Auto	4.5 (0.27)	0.3 (0.40)	2.7 (0.31)	2.9 (0.41)	-4.4 (0.74)	21.3 (0.10)	28.5 (0.05)	-5.5 (0.82)	29.1 (0.03)
	KNN	4.0 (0.19)	1.8 (0.36)	3.3 (0.24)	3.0 (0.31)	-3.0 (0.66)	22.3 (0.10)	24.9 (0.04)	-4.9 (0.83)	28.4 (0.02)
	LR	4.4 (0.18)	1.9 (0.48)	3.8 (0.24)	3.5 (0.30)	-4.5 (0.73)	22.2 (0.11)	25.1 (0.03)	-4.4 (0.84)	29.0 (0.02)
	NB	4.2 (0.36)	4.1 (0.35)	4.3 (0.35)	2.0 (0.51)	-4.3 (0.73)	20.6 (0.08)	21.1 (0.09)	-6.2 (0.78)	28.7 (0.05)
	RF	4.5 (0.20)	2.9 (0.37)	3.7 (0.24)	3.4 (0.32)	-4.6 (0.75)	22.2 (0.10)	23.2 (0.04)	-5.0 (0.83)	29.1 (0.02)
	SVM	4.7 (0.12)	0.2 (0.51)	3.7 (0.17)	4.0 (0.20)	-1.9 (0.67)	22.7 (0.11)	28.8 (0.02)	-4.2 (0.85)	29.3 (0.01)
IND	Auto	1.2 (0.50)	-7.5 (0.53)	-3.9 (0.47)	-2.3 (0.59)	-5.1 (0.86)	63.8 (0.03)	126.3 (0.04)	-8.0 (0.89)	70.0 (0.04)
	KNN	10.2 (0.15)	-1.8 (0.36)	6.0 (0.18)	6.7 (0.22)	-6.2 (0.80)	62.9 (0.06)	81.5 (0.01)	-6.3 (0.91)	70.6 (0.01)
	LR	12.4 (0.32)	-5.0 (0.57)	5.2 (0.36)	6.7 (0.43)	-4.3 (0.82)	54.9 (0.05)	102.0 (0.02)	-6.6 (0.91)	96.0 (0.03)
	NB	5.5 (0.64)	22.7 (0.20)	19.6 (0.27)	-4.0 (0.57)	-4.9 (0.75)	45.1 (0.02)	36.5 (0.26)	-13.6 (0.67)	78.2 (0.05)
	RF	-11.9 (0.62)	-12.6 (0.67)	-13.3 (0.61)	-10.7 (0.70)	-4.9 (0.88)	150.0 (0.03)	162.1 (0.02)	-7.5 (0.91)	42.5 (0.05)
	SVM	7.5 (0.17)	-8.0 (0.56)	2.6 (0.23)	3.6 (0.25)	88.6 (0.77)	62.4 (0.06)	83.7 (0.01)	-5.6 (0.92)	69.9 (0.02)
JIRA	Auto	11.8 (0.38)	8.5 (0.44)	10.2 (0.35)	6.0 (0.50)	-3.4 (0.79)	51.5 (0.07)	62.6 (0.07)	-11.5 (0.82)	85.8 (0.05)
	KNN	16.0 (0.32)	6.4 (0.44)	12.6 (0.31)	10.5 (0.43)	-2.0 (0.72)	52.8 (0.07)	56.3 (0.06)	-11.0 (0.83)	93.4 (0.04)
	LR	12.6 (0.34)	11.4 (0.45)	15.0 (0.30)	8.9 (0.42)	-3.0 (0.76)	55.8 (0.07)	68.4 (0.09)	-9.2 (0.80)	87.7 (0.04)
	NB	8.4 (0.54)	16.2 (0.31)	13.5 (0.34)	0.3 (0.58)	-4.2 (0.78)	51.1 (0.05)	40.0 (0.17)	-15.0 (0.72)	78.8 (0.06)
	RF	16.7 (0.32)	9.6 (0.44)	13.4 (0.32)	10.9 (0.43)	-4.3 (0.80)	54.0 (0.07)	53.7 (0.05)	-11.1 (0.83)	97.9 (0.04)
	SVM	15.4 (0.20)	2.2 (0.51)	12.8 (0.23)	11.7 (0.29)	-1.1 (0.71)	57.4 (0.08)	68.8(0.03)	-9.7 (0.86)	86.8 (0.03)
MA	Auto	0.5 (0.72)	-14.4 (0.76)	-8.6 (0.72)	-9.5 (0.80)	-12.8 (0.93)	10.5 (0.05)	191.5 (0.04)	-18.5 (0.79)	72.9 (0.12)
	KNN	4.9 (0.38)	-15.0 (0.66)	-5.3 (0.45)	-5.3 (0.51)	-13.9 (0.87)	33.9 (0.09)	93.1 (0.04)	-15.4 (0.79)	72.6 (0.08)
	LR	25.6 (0.38)	-6.4 (0.68)	10.0 (0.47)	9.8 (0.53)	-10.2 (0.84)	14.3 (0.10)	92.1 (0.04)	-15.2 (0.79)	128.5 (0.07)
	NB	13.8 (0.55)	17.3 (0.37)	16.5 (0.39)	-3.2 (0.60)	-10.1 (0.77)	11.3 (0.08)	40.5 (0.16)	-21.6 (0.67)	97.7 (0.09)
	RF	-14.7 (0.86)	-19.0 (0.85)	-18.3 (0.84)	-16.4 (0.89)	-13.8 (0.96)	160.8 (0.03)	276.9 (0.02)	-17.4 (0.80)	44.2 (0.14)
	SVM	19.0 (0.31)	-17.0 (0.80)	3.1 (0.43)	5.5 (0.45)	-9.9 (0.86)	29.1 (0.11)	211.0 (0.02)	-14.2 (0.81)	94.0 (0.06)
Met	Auto	12.6 (0.46)	3.5 (0.51)	9.2 (0.41)	3.4 (0.45)	-0.2 (0.67)	7.9 (0.19)	17.1 (0.15)	-14.4 (0.50)	29.2 (0.17)
	KNN	13.5 (0.41)	7.0 (0.52)	12.1 (0.37)	8.2 (0.42)	-0.9 (0.63)	13.1 (0.21)	6.3 (0.14)	-11.2 (0.51)	33.8 (0.14)
	LR	12.9 (0.42)	0.9 (0.57)	10.7 (0.38)	7.3 (0.41)	2.2 (0.65)	11.9 (0.21)	33.4 (0.14)	-12.1 (0.50)	33.4 (0.15)
	NB	11.8 (0.43)	5.2 (0.52)	10.9 (0.39)	7.6 (0.48)	1.1 (0.68)	9.5 (0.22)	14.3 (0.13)	-10.1 (0.52)	31.7 (0.14)
	RF	10.7 (0.45)	6.1 (0.51)	7.9 (0.40)	0.1 (0.46)	-0.6 (0.67)	7.7 (0.20)	16.9 (0.14)	-15.7 (0.51)	31.7 (0.16)
	SVM	12.8 (0.38)	0.6 (0.58)	10.5 (0.35)	6.2 (0.34)	5.6 (0.64)	10.0 (0.21)	21.2 (0.13)	-12.3 (0.52)	33.9 (0.14)
Range		-14.7 ~ 25.6	-19 ~ 22.7	-18.3 ~ 19.6	-16.4 ~ 11.7	-13.9 ~ 88.6	7.7 ~ 160.8	6.3 ~ 276.9	-21.6 ~ -4.2	28.4 ~ 128.5

*: A blue cell indicates values increasing after removing unchanged modules from the target version while a red cell indicates values decreasing after that. Besides the performance delta, the performance value on the original testing set is also reported in parentheses. The darker the color, the larger the absolute value of the performance delta. ECL, IND, JIRA, MA, Met refer to ECLIPSE-2007, IND-JLMIV+R-2020, JIRA-RA-2019, MA-SZZ-2020, and Metrics-Repo-2010, respectively. TP/N, FP/N, TN/N, and FN/N refer to the relative TP, FP, TN, and FN to the total number of instances N in the testing set.

TABLE 5 The Benjamini-Hochberg corrected (and effect size) from the Wilcoxon signed-rank testbetween performance on the target version with and without unchanged data.

Indicator	Model	Auto	KNN	LR	NB	RF	SVM
recall	IND-JLMIV+R-2020	0.389 (n)	**0.000 (s)**	**0.000 (m)**	**0.000 (s)**	**0.000 (l)**	**0.000 (s)**
	JIRA-RA-2019	**0.000 (l)**	**0.000 (l)**	**0.002 (l)**	**0.001 (l)**	**0.008 (l)**	**0.000 (l)**
	MA-SZZ-2020	0.081 (s)	**0.022 (m)**	**0.000 (l)**	**0.014 (m)**	**0.000 (l)**	**0.002 (m)**
	Metrics-Repo-2010	**0.001 (l)**	**0.002 (l)**	**0.002 (l)**	**0.001 (l)**	**0.001 (l)**	**0.001 (l)**
precision	IND-JLMIV+R-2020	**0.000 (m)**	0.366 (n)	**0.006 (s)**	**0.000 (l)**	**0.000 (l)**	**0.000 (m)**
	JIRA-RA-2019	**0.001 (l)**	**0.001 (l)**	**0.006 (l)**	**0.000 (l)**	**0.001 (l)**	**0.048 (m)**
	MA-SZZ-2020	**0.001 (l)**	0.082 (s)	0.715 (n)	**0.004 (m)**	**0.000 (l)**	**0.011 (m)**
	Metrics-Repo-2010	0.120 (s)	**0.004 (l)**	0.542 (s)	**0.018 (m)**	0.105 (m)	0.792 (n)
F1	IND-JLMIV+R-2020	**0.001 (s)**	**0.000 (s)**	**0.000 (s)**	**0.000 (l)**	**0.000 (l)**	**0.001 (s)**
	JIRA-RA-2019	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**	**0.002 (l)**	**0.000 (l)**
	MA-SZZ-2020	0.433 (s)	0.405 (s)	**0.003 (m)**	**0.004 (m)**	**0.000 (l)**	**0.048 (m)**
	Metrics-Repo-2010	**0.002 (l)**	**0.001 (l)**	**0.004 (l)**	**0.001 (l)**	**0.005 (l)**	**0.003 (l)**
G1	IND-JLMIV+R-2020	**0.037 (s)**	**0.000 (s)**	**0.000 (s)**	**0.000 (s)**	**0.000 (l)**	**0.000 (s)**
	JIRA-RA-2019	0.120 (m)	**0.037 (m)**	**0.028 (m)**	0.263 (s)	0.244 (s)	**0.009 (l)**
	MA-SZZ-2020	0.067 (s)	0.542 (s)	**0.002 (m)**	0.843 (n)	**0.000 (l)**	**0.031 (m)**
	Metrics-Repo-2010	0.943 (n)	0.606 (n)	0.716 (n)	**0.026 (m)**	0.792 (n)	0.645 (n)
AUC	IND-JLMIV+R-2020	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**	**0.000 (l)**
	JIRA-RA-2019	**0.004 (l)**	**0.034 (m)**	**0.034 (m)**	**0.006 (l)**	**0.004 (l)**	0.300 (s)
	MA-SZZ-2020	**0.000 (l)**	**0.000 (l)**	0.129 (s)	0.295 (s)	**0.000 (l)**	0.243 (s)
	Metrics-Repo-2010	0.385 (s)	0.480 (s)	0.542 (s)	0.762 (n)	0.311 (s)	0.989 (n)

*: The p-values on the ECLIPSE-2007 data set are not reported since the number of observations in the Eclipse dataset is too small to calculate a meaningful p-value. Cells in bold denote "p-value < 0.05 and effect size is not "negligible". Note that the reported p-values are Benjamini-Hochberg corrected p-values, aimed at controlling the family-wise error rate across multiple tests.

unchanged data from the target versions. However, there are a few exceptions. Specifically, for RF under IND-JLMIV+R-2020 and MA-SZZ-2020, as well as Auto under MA-SZZ-2020, their performances show a noticeable decrease across all indicators.

What is the underlying cause for the aforementioned observations? It is important to note that the deletion of certain instances in the target version does not affect the prediction results of the remaining instances. This is because each instance in the test set is treated independently, and the classifier's prediction for each instance should solely rely on its individual features rather than being influenced by other instances. Hence, after removing unchanged modules from the target version, the prediction results for the remaining modules should remain unaffected. Therefore, any changes observed in a model's prediction performance after the removal of unchanged modules can be attributed solely to the modules that were removed and not due to alterations in the model's predictions for the remaining instances.

For instance, consider precision. Assume that there is a model operating on the original target version, and its confusion matrix consists of TP, FP, TN, and FN. Within this scenario, there are unchanged modules predicted as positive, with a total count of n, out of which d represents the number of true positive unchanged modules. In this context, we define precision for this model on the original target version as "prec" and on the target version without unchanged modules as "prec'":

$$\text{prec} = \text{TP}/(\text{TP}+\text{FP})$$

$$\text{prec}' = (\text{TP}-\text{d})/(\text{TP}+\text{FP}-\text{n})$$

The condition for prec > prec' can be satisfied if and only if the following condition is satisfied:

$$\text{prec} < \text{d}/\text{n}$$

This pattern can be generalized to other threshold-dependent performance indicators, such as recall, F1, and G1, which are based on a confusion matrix. For instance, if the recall of a model on unchanged modules is higher than the recall of the model on all modules in the original target

release, then the recall of the model decreases after removing unchanged modules. In essence, a positive performance delta (caused by pessimistic estimation due to unchanged modules) suggests that a WPDP model performs better at predicting changed modules compared to unchanged modules in the target release, based on a specific performance indicator. Conversely, a negative performance delta (caused by optimistic estimation due to unchanged modules) indicates that a WPDP model performs better at predicting unchanged modules compared to changed modules in the target release, based on a specific performance indicator.

According to the observations presented in Table 4, it is evident that on datasets like ECLIPSE-2007, JIRA-RA-2019, and Metrics-Repo-2010, models generally exhibit poorer performance in predicting unchanged modules compared to changed modules when considering threshold-dependent indicators such as precision, recall, F1, and G1. However, on datasets like MA-SZZ-2020 and IND-JLMIV+R-2020, certain models (RF and Auto) tend to perform better at predicting unchanged modules than changed modules. This finding can be attributed to the specific characteristics of these datasets, particularly the difficulty in correctly predicting unchanged modules. Further analysis and discussion on this topic will be provided in Section 6.1.

> **Conclusion.** Data duplication significantly affects reported performance values for each learner. Optimistic estimation or pessimistic estimation occurs due to the model's better (or worse) performance in predicting unchanged modules compared with all modules in the original target release.

5.3 | RQ3: how does data duplication affect comparison results between multiple WPDP models?

Table 6 presents the ranking of models' performances, as determined by a two-level NPSK test. The "dup" column displays the rankings of models' performances on the original target version (including unchanged modules), while the "w/o dup" column shows the rankings of models' performances on the target version without unchanged modules.

From Table 6, we make the following observations.

- *Observation 1: overall, the presence of unchanged data does not significantly influence the ranking of multiple modules*. The p-values resulting from the Wilcoxon signed rank test, measuring the discrepancy in one-level NPSK rankings when testing with "dup" or "w/o dup" target releases, are as follows: 0.04 for recall, 0.48 for precision, 0.17 for F_1, 0.03 for G_1, and 0.39 for AUC. Consequently, the disparity in ranking, specifically concerning precision, F_1, and AUC, is insignificant after the elimination of unchanged modules. For instance, Table 6 demonstrates that both Auto and RF exhibit the highest F_1 scores in both "dup" and "w/o dup" scenarios. Similarly, RF achieves the highest AUC scores in both cases. SVM, while excelling in precision for both "dup" and "w/o dup," ranks poorly compared to other models in terms of additional performance indicators.
- *Observation 2: Despite being in the forefront of rankings, AutoML technology does not exhibit a significant performance advantage over traditional machine learning approaches in the context of WPDP*. In Table 6, the average rankings across all performance indicators for Auto, KNN, LR, NB, RF, and SVM are as follows: under "dup," Auto ranks at 1.80, KNN at 3.20, LR at 2.80, NB at 2.20, RF at 1.60, and SVM at 3.60; under "w/o dup," Auto remains at 1.80, KNN increases to 3.80, LR remains at 2.80, NB increases to 2.40, RF increases to 1.80, and SVM increases to 3.80. We can see RF is a strong machine learning classifier with a better average ranking under "dup" (1.60) and the same ranking under "w/o dup" (1.80) compared to Auto. Considering the robustness of models on unseen data (i.e., modules that remain in the target releases after removing unchanged modules), the performance of Auto, LR, and RF demonstrate similar results. Specifically, Auto and LR are the only two models whose average ranking does not decrease after eliminating data duplication, and RF is on the same rank in 4 out of 5 metrics, which is also stable.
- *Observation 3: comparing multiple WPDP models, it is more likely for data duplication to introduce bias in the assessment of recall and G1 performance indicators*. Specifically, only the p-values associated with recall and G1 are lower than 0.05, indicating that the rankings of model

TABLE 6 Ranking of models by two-level NPSK test.

Indicator	Recall		Precision		F_1		G_1		AUC	
diffToTarget	Dup	w/o dup	Dup	w/o dup	Dup	w/o dup	Dup	w/o dup	Dup	w/o dup
Auto	2	2	2	3	1	1	2	2	2	1
KNN	3	4	3	4	3	4	3	3	4	4
LR	3	4	2	2	2	3	4	3	3	2
NB	1	1	4	5	2	2	1	1	3	3
RF	2	3	2	2	1	1	2	2	1	1
SVM	4	5	1	1	4	5	5	4	4	4

performances significantly differ when testing on "dup" or "w/o dup" target releases in terms of recall and G1. On the other hand, the discrepancies observed in other performance indicators are considered insignificant.

Conclusion. When ranking multiple WPDP models based on their prediction performance, the influence of removing unchanged instances is not substantial. However, the removal of unchanged instances does have a slight effect on the selection of models with superior generalization abilities.

6 | DISCUSSION

In this section, we analyze the reasons for performance optimistic estimation and performance pessimistic estimation due to unchanged modules and examine the impact of resampling on WPDP models' generalization for unseen modules in the target release.

6.1 | What factors influence whether the presence of data duplication leads to an optimistic estimation or a pessimistic estimation of a model's performance?

In Section 5.2, it was observed that the performances of WPDP models were pessimistically estimated in some defect datasets, while they were optimistically estimated in others. We previously mentioned that if a WPDP model excels at predicting unchanged modules compared to other modules in the target version, data duplication can lead to an optimistic estimation in its performance evaluation. Conversely, if a WPDP model performs poorly in predicting unchanged modules compared to other modules in the target version, data duplication can result in a pessimistic estimation. However, it remains to be determined whether classification models are inherently good or bad at predicting unchanged modules. The answer to this question can explain the trend observed in Table 4, where models were generally pessimistically estimated due to data duplication in ECLIPSE-2007, JIRA-RA-2019, and Metrics-Repo-2010, while they were generally optimistically estimated due to data duplication in IND-JLMIV+R-2020 and MA-SZZ-2020.

To address this question, we classified unchanged modules into four types based on their labeling in the target and source versions:

- BB (Buggy in target version and Buggy in source version): Unchanged instances labeled as buggy in both the target and source versions.
- CB (Clean in target version and Buggy in source version): Unchanged instances labeled as clean in the target version but buggy in the source version.
- CC (Clean in both target and source versions): Unchanged instances labeled as clean in both the target and source versions.
- BC (Buggy in target version and Clean in source version): Unchanged instances labeled as buggy in the target version but clean in the source version.

Figure 5 illustrates the ratio of CB to the combined count of CB and BB, as well as the ratio of BB to the combined count of CB and BB. Figure 6 depicts the ratio of BC to the combined count of BC and CC, and the ratio of CC to the combined count of BC and CC. Among these types, BB and CC represent unchanged modules with consistent labels across source and target versions. These modules are relatively easier to predict as supervised models have learned their correct labels from the source version. On the other hand, CB and BC refer to unchanged modules with inconsistent labels in the target version compared to their labels in the source version. These modules pose a challenge for prediction. Due to their identical source codes, the metrics extracted from their source code tend to be the same in both versions. When predicting the labels of these modules in the target version, models often tend to assign the same label as in the source version, which is different from their label in the target version. In conclusion, when the ratio of inconsistent unchanged modules (CB and BC) to all unchanged modules is high, it tends to result in a pessimistic estimation of a model's prediction performance.

As observed in Figure 6, the ratios of inconsistent labels (BC and CC) are generally very low across all defect datasets, indicating that inconsistent labels are rarely observed in negative unchanged modules. In Figure 5, the values of BB/(CB + BB) in the IND-JLMIV+R-2020 and MA-SZZ-2020 datasets are indeed extremely high. These datasets are newly proposed and place emphasis on dataset quality, including the consistency of labels for identical modules in consecutive versions. As a result, the unchanged modules in these datasets are easier to predict, and removing unchanged modules leads to a decrease in WPDP model performance, as shown in Table 4. Conversely, the correct values to consider are CB/(CB + BB) in the ECLIPSE-2007, JIRA-RA-2019, and Metrics-Repo-2010 datasets, which are extremely high. This indicates a high ratio of inconsistent labels in these older defect datasets. Consequently, the unchanged modules in these datasets are more challenging to predict. In such cases, removing unchanged modules leads to an increase in WPDP model performance in Table 4. To summarize, the presence of high values in BB/(CB + BB) in the newly proposed datasets (IND-JLMIV+R-2020 and MA-SZZ-2020) and high values in CB/(CB + BB) in the older defect datasets (ECLIPSE-2007, JIRA-RA-2019, and Metrics-Repo-2010) explain the observed trends in Table 4.

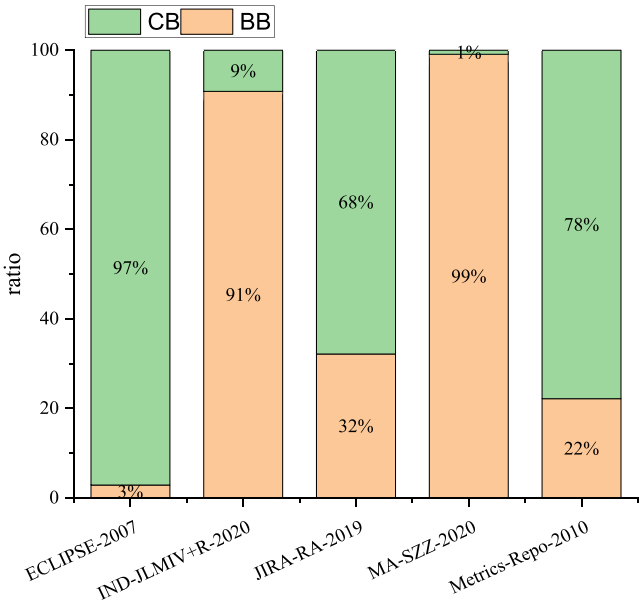

FIGURE 5 For all unchanged modules that are labeled as buggy in the source version, the ratio of modules whose labels are inconsistent (CB) or consistent (BB).

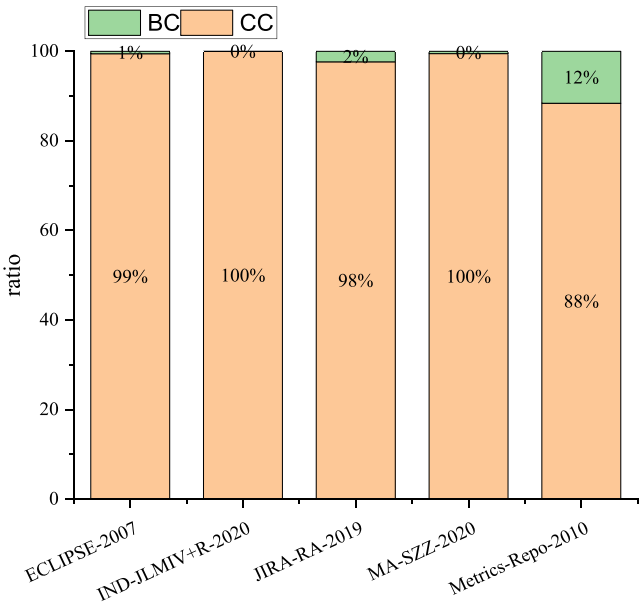

FIGURE 6 For all unchanged modules that are labeled as clean in the source version, the ratio of modules whose labels are inconsistent (BC) or consistent (CC).

Conclusion. Unchanged modules in the target version with consistent labels (compared to the source version) can potentially lead to optimistic estimation when evaluating WPDP models. Unchanged modules in the target version with inconsistent labels (compared to the source version) can potentially lead to pessimistic estimation when evaluating WPDP modules.

6.2 | Can the resampling strategy help WPDP models become more generalizable to unseen modules in target versions?

In Section 6.1, we observed that SDP models can be overly optimistic evaluated when learning unchanged modules from the source version, leading to inadequate generalization for predicting unseen modules in the target version. The resampling strategy, commonly employed in SDP to

address imbalanced datasets, has been shown to enhance generalization performance.[54–57] Consequently, it is reasonable to inquire whether the resampling strategy can aid WPDP models in achieving better generalization to unseen modules (changed modules) in target versions.

To investigate the aforementioned issue, we employ SMOTE (Synthetic Minority Oversampling Technique),[58] implemented using the imbalanced-learn Python library, to balance the modules in the source versions. SMOTE is specifically designed to tackle class imbalance by generating synthetic examples of the minority class, thereby increasing its representation in the dataset. It is a widely used resampling strategy in SDP studies.[59–61] In our case, we apply SMOTE to achieve a 1:1 ratio between buggy and clean modules in the source versions. Subsequently, we evaluate whether SMOTE can enhance the classifier's performance on the original target version, which lacks unchanged modules. A positive outcome would indicate that the resampling strategy improves the generalization of SDP models to these new modules in target versions.

Table 7 reports the average performance metrics of classifiers, including the performance change ratio (delta), the p-value (p), and the effect size (eff) when comparing classifiers with and without SMOTE across all target versions in the five datasets. For a detailed summary of each dataset, please refer to our replication kit.[42]

From Table 7, we make the following observations.

- In the case of all six classifiers, SMOTE demonstrates a statistically significant improvement in their generalization. For instance, using SMOTE, the mean recall of KNN on unseen data increases from 0.22 to 0.64 (a boost of 186%) with a "large" effect size. Similarly, the mean F_1 and G_1 scores witness increases of 53% and 113% respectively, also accompanied by "large" effect sizes.
- Classifiers such as NB and RF, which initially performed well on unseen modules (in the target version without unchanged modules), demonstrated relatively minor improvements when SMOTE was applied. Conversely, classifiers like KNN, LR, and SVM, which initially had poor performance on unseen modules, exhibited substantial improvements with SMOTE. For example, the recall improvement from "without unchanged modules" to "SMOTE + without unchanged modules" was only 3% for NB, whereas it reached 153% for SVM.

Conclusion. The application of resampling strategies to training versions significantly enhances the generalization of WPDP models to unseen modules in target versions. In practical evaluations where models are tested on target versions without unchanged modules, resampling strategies are highly recommended for improved performance.

7 | COMPARISON WITH RELATED WORK

In the following, we conduct a comparison between our work and the existing literature[62] pertaining to our research topic. We aim to provide a comprehensive analysis that highlights the similarities, differences, and contributions of our study in relation to the prior research. In their study, Shepperd et al[62] examined the preprocessing techniques applied to the NASA defect dataset to ensure data quality. This involved the removal of "identical cases" and "inconsistent cases," which may bear a resemblance to the concept of "unchanged modules" discussed in our work. However, it is important to note that the notion of "unchanged modules" in our study differs from their concepts in multiple ways.

The primary distinction between our definition of "unchanged modules" and Shepperd et al's concept of "identical cases" or "inconsistent cases" lies in the granularity of the comparison. In our study, we define "unchanged modules" as modules within different versions that share the same "non-blank executable source code." This definition focuses specifically on the similarity of the source code at the module level. In contrast, Shepperd et al's definition of "identical cases" refers to items within a dataset that possess identical feature values and class labels. Similarly, "inconsistent cases" refer to items within the dataset that exhibit identical feature values but have differing class labels. It is important to note that their definitions operate at a different level of granularity, focusing on feature values and class labels rather than the source code of individual modules. Consequently, it is essential to acknowledge that modules identified as "identical cases" or "inconsistent cases" in their study may not necessarily align with our definition of "unchanged modules" based on source code similarity.

Another distinction between our work and Shepperd et al's study[62] is in the interpretation of "duplicate" or "identical." In our research, an "unchanged module" refers to a module in the target version that exhibits duplication with a module in the source version. On the other hand, in,[62] "identical" pertains to an item within the NASA defect dataset that possesses another item with identical feature values. Therefore, the term "unchanged" in our study specifically pertains to module-level duplication between versions, while the term "identical" in[62] relates to items within the dataset having identical feature values. These distinctions highlight the different contexts in which the terms are applied and underscore the divergent interpretations of "unchanged" and "identical" between the two studies.

Furthermore, there is a distinction in the research subjects between Shepperd et al's study[62] and our research. They focused on analyzing the NASA defect dataset and related studies up until June 2012. In contrast, our investigation encompasses different cross-version defect datasets, namely ECLIPSE-2007,[35] Metrics-Repo-2010,[36–38] JIRA-RA-2019,[39] IND-JLMIV+R-2020,[40] and MA-SZZ-2020.[41] These datasets offer a broader scope and represent diverse software projects for our analysis of unchanged modules' prevalence and impact on performance evaluations.

In summary, Shepperd et al's study[62] primarily concentrated on analyzing data quality in the NASA defect dataset and made substantial contributions to its improvement. Their research emphasized the significance of data quality and influenced subsequent studies in this area. In

TABLE 7 The mean performance difference across all target versions in the five datasets between classifiers trained with and without SMOTE preprocessing on source versions.

Model	Indicator	Dup	Smote + dup	delta1	p (eff)1	w/o dup	Smote + w/o dup	p (eff)2	delta2
Auto	recall	0.50	0.65	29%	**0.000 (l)**	0.51	0.61	**0.000 (l)**	19%
	precision	0.54	0.58	7%	**0.000 (s)**	0.50	0.52	0.857 (n)	3%
	F_1	0.48	0.58	21%	**0.000 (l)**	0.46	0.51	**0.000 (m)**	13%
	G_1	0.59	0.71	20%	**0.000 (l)**	0.57	0.66	**0.000 (l)**	15%
	AUC	0.85	0.86	1%	**0.000 (m)**	0.80	0.80	**0.002 (s)**	1%
KNN	recall	0.20	0.65	224%	**0.000 (l)**	0.22	0.64	**0.000 (l)**	186%
	precision	0.41	0.28	-30%	**0.000 (l)**	0.39	0.30	**0.000 (m)**	-22%
	F_1	0.24	0.37	56%	**0.000 (l)**	0.25	0.38	**0.000 (l)**	53%
	G_1	0.28	0.67	142%	**0.000 (l)**	0.29	0.62	**0.000 (l)**	113%
	AUC	0.79	0.81	3%	**0.000 (l)**	0.74	0.76	**0.000 (l)**	4%
LR	recall	0.33	0.57	74%	**0.000 (l)**	0.36	0.58	**0.000 (l)**	60%
	precision	0.57	0.33	-42%	**0.000 (l)**	0.54	0.35	**0.000 (l)**	-35%
	F_1	0.37	0.38	3%	0.039 (n)	0.39	0.40	0.148 (n)	2%
	G_1	0.43	0.64	49%	**0.000 (l)**	0.45	0.62	**0.000 (l)**	36%
	AUC	0.80	0.77	-3%	**0.000 (l)**	0.76	0.73	**0.000 (l)**	-4%
NB	recall	0.60	0.63	5%	**0.000 (s)**	0.63	0.65	**0.000 (s)**	3%
	precision	0.25	0.23	-10%	**0.000 (l)**	0.28	0.26	**0.000 (l)**	-9%
	F_1	0.29	0.28	-6%	**0.000 (m)**	0.33	0.32	**0.000 (m)**	-5%
	G_1	0.56	0.57	0%	0.039 (n)	0.55	0.53	0.248 (n)	-2%
	AUC	0.75	0.72	-4%	**0.000 (l)**	0.71	0.68	**0.000 (l)**	-4%
RF	recall	0.60	0.66	9%	**0.000 (l)**	0.53	0.60	**0.000 (l)**	13%
	precision	0.66	0.61	-7%	**0.000 (l)**	0.58	0.53	**0.000 (l)**	-9%
	F_1	0.59	0.60	1%	0.238 (n)	0.51	0.52	**0.004 (s)**	3%
	G_1	0.67	0.72	7%	**0.000 (l)**	0.60	0.66	**0.000 (l)**	10%
	AUC	0.87	0.87	0%	0.367 (n)	0.82	0.81	**0.000 (s)**	-1%
SVM	recall	0.20	0.56	183%	**0.000 (l)**	0.22	0.57	**0.000 (l)**	153%
	precision	0.57	0.33	-43%	**0.000 (l)**	0.51	0.35	**0.000 (l)**	-32%
	F_1	0.26	0.38	48%	**0.000 (l)**	0.27	0.39	**0.000 (l)**	44%
	G_1	0.28	0.62	125%	**0.000 (l)**	0.29	0.59	**0.000 (l)**	101%
	AUC	0.76	0.78	3%	**0.003 (s)**	0.73	0.74	0.148 (n)	2%

*: The column "dup" indicates the classifier was tested on target versions with unchanged modules, "smote + dup" indicates the classifier was trained on training data with SMOTE preprocessing and tested on target versions with unchanged modules, "w/o dup" indicates the classifier was tested on target versions without unchanged modules, "smote + w/o dup" indicates the classifier was trained on training data with SMOTE preprocessing and test on target versions without unchanged modules. The column "delta1" indicates the ratio of performance difference between "smote + dup" and "dup" in relation to the performance of "dup". The column "delta2" indicates the ratio of performance difference between "smote + w/o dup" and "w/o dup" in relation to the performance value of "w/o dup". The column "p (eff)1" denote the Bonferroni-adjusted p-value of the Wilcoxon signed-rank test and the magnitude of effect size between "dup" and "smote + dup". The column "p (eff)2" denote the Bonferroni-adjusted p-value of the Wilcoxon signed-rank test and the magnitude of effect size between "w/o dup" and "smote + w/o dup". A blue "delta" denotes that delta > 0, indicating that the performance of a classifier with SMOTE is better than without SMOTE. A red "delta" denotes that delta < 0. Bold "p (eff)" indicates that the p-value is less than 0.05, and the magnitude of effect size is not negligible. Note that the reported p-values are Benjamini-Hochberg corrected p-values, aimed at controlling the family-wise error rate across multiple tests.

contrast, our work centers on enhancing performance evaluation in SDP, specifically examining the influence of unchanged modules on WPDP model evaluation. Our study differs from theirs in terms of research objectives, granularity, definitions of concepts, and subject datasets.

8 | THREATS TO VALIDITY

In this section, we discuss the potential threats to the validity of our study that need to be considered in interpreting the findings of our study.

8.1 | Construct validity

Construct validity refers to the extent to which the chosen measures or operationalizations accurately represent the constructs or concepts being studied. The primary concern regarding construct validity in our study lies in the accuracy of our method for identifying unchanged modules, particularly with respect to developers' comprehension of whether a module is indeed unchanged. Our current approach, as outlined in Section 3, considers functions with distinct names but identical code context as "changed." However, developers' understanding of module duplication across different versions may differ from our interpretation. Nonetheless, we posit that such discrepancies are infrequent. Our criteria, which define "unchanged modules" as those sharing the same module name and non-blank executable source code, should hold true for the majority of modules and remain valid.

8.2 | Internal validity

Internal validity refers to the extent to which a cause-and-effect relationship can be established between the independent and dependent variables in a study. One potential risk to internal validity is the improper implementation of our code. To address this concern, we took measures to mitigate the risk. Specifically, we utilized the pre-existing code analysis tool Understand (https://scitools.com/) for identifying duplication, employed the open-source machine learning Python library scikit-learn for training and testing classification models, and utilized the Python library imbalanced-learn to apply the SMOTE technique. Additionally, for transparency and reproducibility, both our code and data are available.[42] These steps were taken to minimize the potential threat to internal validity arising from inaccurate code implementation.

8.3 | External validity

External validity refers to the extent to which research findings can be generalized to contexts beyond the specific study. In our study, the external validity relates to the generalizability of the findings and conclusions derived from the analysis of the five datasets to other software projects. It is important to note that the five datasets used in our study encompass a variety of software projects, exhibiting diversity in terms of project types, sizes, domains, metrics, and data collection processes.[29] This diversity enhances the likelihood of generalizability to a broader range of projects. Additionally, we have made available a replication kit[32] that allows researchers to validate our findings on other software projects, further facilitating the assessment of external validity in different contexts.

9 | CONCLUSION

In this study, we examine the occurrence of unchanged modules in cross-version defect datasets, uncovering their impact on biased performance evaluations of WPDP (Within-Project Defect Prediction) models. We also investigate the underlying reasons behind this performance bias caused by unchanged modules. To achieve this, we conduct comparative experiments using five defect datasets, comprising 481 pairs of consecutive versions as training and test sets. Through the utilization of statistical tests such as the Wilcoxon signed rank test and effect size analysis, we identify significant differences in performance between target versions with and without unchanged data. These findings underscore the importance of removing unchanged modules from test data during WPDP performance evaluations. Consequently, we strongly recommend that future WPDP studies consider the removal of unchanged modules from target versions when assessing model performance.

In future work, we aim to broaden our experiment by including a wider range of projects. This expansion will enhance the generalizability and robustness of our findings. It will also help validate our results and contribute to the development of more accurate defect prediction models. Moreover, we may focus on the investigation of performance indicators with robust measurement of the performance delta of SDP models after removing unchanged modules from test sets in the future. Besides, other strategies to detect unchanged modules can be investigated other than our proposed lexical-based approach, for example, bytecode granularity unchanged module detection.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in wpdp-effectiveness at https://github.com/liu906/wpdp-effectiveness.

ORCID

Xutong Liu https://orcid.org/0000-0002-3831-5505
Yuanqing Mei https://orcid.org/0000-0003-3122-8887
Yuming Zhou https://orcid.org/0000-0002-4645-2526

REFERENCES

1. Trautsch A, Herbold S, Grabowski J. Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction. In: *2020 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE; 2020:127-138.
2. Ni C, Wang W, Yang K, Xia X, Liu K and Lo D, The best of both worlds: Integrating semantic features with expert features for defect prediction and localization, in: Proc. Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery; 2022: 672-683, numpages = 612.
3. Rahman F, Devanbu P. How, and why, process metrics are better. In: *2013 35th international conference on software engineering (ICSE)*. IEEE; 2013: 432-441.
4. Zhou YM, Yang YB, Lu HM, et al. How far we have progressed in the journey? An examination of cross-project defect prediction. *ACM Trans. Softw. Eng. Methodol.* 2018;27(1):1-51. doi:10.1145/3183339
5. Wang S, Liu TY, Nam J, Tan L. Deep semantic feature learning for software defect prediction. *IEEE Trans. Software Eng.* 2018;46(12):1267-1293. doi:10.1109/TSE.2018.2877612
6. Qu Y, Zheng QH, Chi JL, et al. Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Trans. Software Eng.* 2021;47(2):348-366. doi:10.1109/TSE.2019.2892959
7. Wen M, Wu RX, Cheung SC. How well do change sequences predict defects? Sequence learning from software changes. *IEEE Trans. Software Eng.* 2018;46(11):1155-1175. doi:10.1109/TSE.2018.2876256
8. Li K, Xiang ZL, Chen T, Wang S and Tan KC, Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: An empirical study, in: Proc. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering; 2020: 566-577.
9. Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Trans. Software Eng.* 2019;45(2):111-147. doi:10.1109/TSE.2017.2770124
10. Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. In: *Proc. proceedings of the 40th international conference on software engineering*. Association for Computing Machinery; 2018:1063.
11. Li Z, Zhang H, Jing X-Y, Xie J, Guo M, Ren J. Dssdpp: data selection and sampling based domain programming predictor for cross-project defect prediction. *IEEE Trans. Software Eng.* 2023;49(4):1941-1963. doi:10.1109/TSE.2022.3204589
12. Chen H, Jing X-Y, Li Z, Wu D, Peng Y, Huang Z. An empirical study on heterogeneous defect prediction approaches. *IEEE Trans. Software Eng.* 2021; 47(12):2803-2822. doi:10.1109/TSE.2020.2968520
13. Pachouly J, Ahirrao S, Kotecha K, Selvachandran G, Abraham A. A systematic literature review on software defect prediction using artificial intelligence: datasets, data validation methods, approaches, and tools. *Eng Appl Artif Intel*. 2022;111:104773. doi:10.1016/j.engappai.2022.104773
14. Bhandari K, Kumar K, Sangal AL. Data quality issues in software fault prediction: a systematic literature review. *Artif Intell Rev*. 2023;56(8):7839-7908. doi:10.1007/s10462-022-10371-6
15. Moussa R, Sarro F. On the use of evaluation measures for defect prediction studies. In: *Proc. proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis*. Association for Computing Machinery; 2022:101-113.
16. Amasaki S. Cross-version defect prediction: use historical data, cross-project data, or both? *Empirical Software Eng.* 2020;25(2):1573-1595. doi:10.1007/s10664-019-09777-8
17. Nam J, Kim S. Clami: Defect prediction on unlabeled datasets (t). In: *Proc*. Vol.2015. ASE; 2015:452-463.
18. Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: *Proc. 2016 IEEE/ACM 38th international conference on software engineering (ICSE)*. IEEE; 2016:309-320.
19. Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. In: *Proc. proceedings of the 40th international conference on software engineering*. Association for Computing Machinery; 2018:699.
20. Gong LN, Jiang SJ, Wang RC, Jiang L. Empirical evaluation of the impact of class overlap on software defect prediction. In: *Proc. ASE 2019*. IEEE; 2019:698-709.
21. Gong L, Zhang H, Zhang J, Wei M, Huang Z. A comprehensive investigation of the impact of class overlap on software defect prediction. *IEEE Trans. Software Eng.* 2023;49(4):2440-2458. doi:10.1109/TSE.2022.3220740
22. Cover T, Hart P. Nearest neighbor pattern classification. *IEEE Trans Inf Theory*. 1967;13(1):21-27. doi:10.1109/TIT.1967.1053964
23. Hosmer DW Jr. Lemeshow S and Sturdivant RX. In: *Applied logistic regression*. John Wiley & Sons; 2013. doi:10.1002/9781118548387
24. Lewis DD. Naive (bayes) at forty: The independence assumption in information retrieval. In: *Proc. European conference on machine learning*. Springer; 1998:4-15. doi:10.1007/BFb0026666
25. Breiman L. Random forests. *Mach Learn*. 2001;45(1):5-32. doi:10.1023/A:1010933404324
26. Suykens JA, Vandewalle J. Least squares support vector machine classifiers. *Neural Process Lett*. 1999;9(3):293-300. doi:10.1023/A:1018628609742
27. Kang HJ, Aw KL, Lo D. Detecting false alarms from automatic static analysis tools: How far are we? In: *Proc. proceedings of the 44th international conference on software engineering*. Association for Computing Machinery; 2022:698-709.
28. Allamanis M. The adverse effects of code duplication in machine learning models of code. In: *Proc. proceedings of the 2019 ACM SIGPLAN international symposium on new ideas, new paradigms, and reflections on programming and software*. Association for Computing Machinery; 2019:143-153.
29. Liu S, Guo Z, Li Y, et al. Inconsistent defect labels: Essence, causes, and influence. *IEEE Trans. Software Eng.* 2023;49(2):586-610. doi:10.1109/TSE.2022.3156787
30. Rodríguez-Pérez G, Robles G, Serebrenik A, Zaidman A, Germán DM, Gonzalez-Barahona JM. How bugs are born: a model to identify how bugs are introduced in software components. *Empirical Software Eng*. 2020;25(2):1294-1340. doi:10.1007/s10664-019-09781-y
31. Erik A, Lionel CB, Eivind BJ. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J Syst Softw*. 2010;83(1):2-17. doi:10.1016/j.jss.2009.06.055
32. Wahono RS. A systematic literature review of software defect prediction. *J Software Eng*. 2015;1(1):1-16.
33. Görkem G, Kwabena Ebo B, Ömer K, Önder B, Bedir T. On the use of deep learning in software defect prediction. *J Syst Softw*. 2023;195:111537. doi:10.1016/j.jss.2022.111537
34. Erickson N, Mueller J, Shirkov A, Zhang H, Larroy P, Li M and Smola A, Autogluon-tabular: Robust and accurate automl for structured data. arXiv preprint arXiv:2003.06505. 2020.

35. Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: *Third international workshop on predictor models in software engineering (PROMISE'07: ICSE workshops 2007)*. IEEE; 2007:9-9.
36. Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In: *Proc. proceedings of the 6th international conference on predictive models in software engineering*. Association for Computing Machinery; 2010. Article 9
37. Jureczko M, Spinellis D. Using object-oriented design metrics to predict software defects. Models and methods of system dependability. *Oficyna Wydawnicza Politechniki Wrocławskiej*. 2010;69-81.
38. Menzies T, Caglayan B, Kocaguneli E, Krall J, Peters F and Turhan B, The promise repository of empirical software engineering data, Book The promise repository of empirical software engineering data, Series The promise repository of empirical software engineering data, ed., Editor ed.^eds., June, 2012, pp.
39. Yatish S, Jiarpakdee J, Thongtanunam P, Tantithamthavorn C. Mining software defects: Should we consider affected releases? In: *2019 IEEE/ACM 41st international conference on software engineering (ICSE)*. IEEE; 2019:654-665.
40. Herbold S, Trautsch A, Trautsch F, Ledel B. Problems with szz and features: an empirical study of the state of practice of defect prediction data collection. *Empirical Software Eng.* 2022;27(2):42. doi:10.1007/s10664-021-10092-4
41. da Costa DA, McIntosh S, Shang W, Kulesza U, Coelho R, Hassan AE. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Trans. Software Eng.* 2017;43(7):641-657. doi:10.1109/TSE.2016.2616306
42. Liu X, The replication kit for "the impact of data duplication on within-project defect prediction model evaluation", 2023; https://github.com/liu906/wpdp-effectiveness
43. Jingxiu Y, Martin S. The impact of using biased performance metrics on software defect prediction research. *Inf. Software Technol.* 2021;139:106664.
44. Issam HL, Mohammad A, Lahouari G. Software defect prediction using ensemble learning on selected features. *Inf. Software Technol.* 2015;58:388-402.
45. Wang S, Yao X. Using class imbalance learning for software defect prediction. *IEEE Trans Reliab*. 2013;62(2):434-443. doi:10.1109/TR.2013.2259203
46. Hollander M, Wolfe DA, Chicken E. *Nonparametric statistical methods*. John Wiley & Sons; 2013.
47. Fritz CO, Morris PE, Richler JJ. Effect size estimates: current use, calculations, and interpretation. *J Exp Psychol Gen*. 2012;141(1):2-18. doi:10.1037/a0024338
48. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Software Eng.* 2017;43(1):1-18. doi:10.1109/TSE.2016.2584050
49. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Software Eng.* 2019;45(7):683-711. doi:10.1109/TSE.2018.2794977
50. Li Z, Jing XY, Zhu X, Zhang H, Xu B, Ying S. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Trans. Software Eng.* 2019;45(4):391-411. doi:10.1109/TSE.2017.2780222
51. Li Z, Jing X-Y, Zhu X, Zhang H, Xu B, Ying S. Heterogeneous defect prediction with two-stage ensemble learning. *Autom Software Eng*. 2019;26(3):599-651. doi:10.1007/s10515-019-00259-1
52. Li Z, Zhang H, Jing X-Y, Xie J, Guo M, Ren J. *Dssdpp: data selection and sampling based domain programming predictor for cross-project defect prediction*. IEEE Trans; 2022.
53. Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: *Proc. proceedings of the 30th international conference on software engineering*. Association for Computing Machinery; 2008:181-190. numpages = 110
54. Pelayo L, Dick S. Applying novel resampling strategies to software defect prediction. In: *NAFIPS 2007-2007 annual meeting of the north American fuzzy information processing society*. IEEE; 2007:69-72.
55. Bennin KE, Keung JW, Monden A. On the relative value of data resampling approaches for software defect prediction. *Empirical Software Eng.* 2019;24(2):602-636. doi:10.1007/s10664-018-9633-6
56. Tantithamthavorn C, Hassan AE, Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Trans. Software Eng.* 2020;46(11):1200-1219. doi:10.1109/TSE.2018.2876537
57. Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. Mahakil: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering*. 2018;44(6):534-550. doi:10.1109/TSE.2017.2731766
58. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. Smote: synthetic minority over-sampling technique. *J Artif Intell Res*. 2002;16:321-357. doi:10.1613/jair.953
59. Feng S, Keung J, Yu X, Xiao Y, Zhang M. Investigation on the stability of smote-based oversampling techniques in software defect prediction. *Inf. Software Technol.* 2021;139:106662. doi:10.1016/j.infsof.2021.106662
60. El-Shorbagy SA, El-Gammal WM, Abdelmoez WM. Using smote and heterogeneous stacking in ensemble learning for software defect prediction. In: *Proc. proceedings of the 7th international conference on software and information engineering*. Association for Computing Machinery; 2018:44-47.
61. Limsettho N, Bennin KE, Keung JW, Hata H, Matsumoto K. Cross project defect prediction using class distribution estimation and oversampling. *Inf Software Technol*. 2018;100:87-102. doi:10.1016/j.infsof.2018.04.001
62. Shepperd M, Song Q, Sun Z, Mair C. Data quality: some comments on the nasa software defect datasets. *IEEE Trans Software Eng*. 2013;39(9):1208-1215. doi:10.1109/TSE.2013.11

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Liu X, Zhou Y, Lu Z, et al. Unveiling the impact of unchanged modules across versions on the evaluation of within-project defect prediction models. *J Softw Evol Proc*. 2024;36(12):e2715. https://doi.org/10.1002/smr.2715