Science of Computer Programming 238 (2024) 103164

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Towards a framework for reliable performance evaluation in defect prediction

Xutong Liu, Shiran Liu, Zhaoqiang Guo, Peng Zhang, Yibiao Yang, Huihui Liu, Hongmin Lu, Yanhui Li, Lin Chen, Yuming Zhou *

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, Jiangsu, China

ARTICLE INFO

Keywords:
Defect prediction
Performance evaluation
Prediction models
Software quality assurance (SQA)

ABSTRACT

Enhancing software reliability, dependability, and security requires effective identification and mitigation of defects during early development stages. Software defect prediction (SDP) models have emerged as valuable tools for this purpose. However, there is currently a lack of consensus in evaluating the predictive performance of newly proposed models, which hinders accurate measurement of progress and can lead to misleading conclusions. To tackle this challenge, we present MATTER (a fraMework towArd a consisTenT pErformance compaRison), which aims to provide reliable and consistent performance comparisons for SDP models. MATTER incorporates three key considerations. First, it establishes a global reference point, ONE (glObal baseliNe modEl), which possesses the 3S properties (Simplicity in implementation, Strong predictive ability, and Stable prediction performance), to serve as the baseline for evaluating other models. Second, it proposes using the SQA-effort-aligned threshold setting to ensure fair performance comparisons. Third, it advocates for consistent performance evaluation by adopting a set of core performance indicators that reflect the practical value of prediction models in achieving tangible progress. Through the application of MATTER to the same benchmark data sets, researchers and practitioners can obtain more accurate and meaningful insights into the performance of defect prediction models, thereby facilitating informed decision-making and improving software quality. When evaluating representative SDP models from recent years using MATTER, we surprisingly observed that: none of these models demonstrated a notable enhancement in prediction performance compared to the simple baseline model ONE. In future studies, we strongly recommend the adoption of MATTER to assess the actual usefulness of newly proposed models, promoting reliable scientific progress in defect prediction.

1. Introduction

Defect prediction plays an important role in the development and maintenance of dependable and secure software systems. By utilizing a defect prediction model, it becomes possible to anticipate the probable locations of defects within a software project, such as identifying defect-prone modules like files, classes, or functions. In practical terms, this information is invaluable for software

* Corresponding author.
E-mail addresses: xryu@smail.nju.edu.cn (X. Liu), shiranliu@smail.nju.edu.cn (S. Liu), gzq@smail.nju.edu.cn (Z. Guo), DZ1833034@smail.nju.edu.cn (P. Zhang), yangyibiao@nju.edu.cn (Y. Yang), huihuiliu@nju.edu.cn (H. Liu), hmlu@nju.edu.cn (H. Lu), yanhuili@nju.edu.cn (Y. Li), lchen@nju.edu.cn (L. Chen), zhouyuming@nju.edu.cn (Y. Zhou).

https://doi.org/10.1016/j.scico.2024.103164
Received 26 January 2024; Received in revised form 9 April 2024; Accepted 5 June 2024

Available online 12 June 2024

quality assurance activities. On the one hand, it aids in prioritizing modules for inspection or testing. By focusing on highly defect-prone modules, defects can be discovered at an early stage in the software project. On the other hand, it assists in effectively allocating inspection and testing efforts. Concentrating resources on modules with a high likelihood of defects maximizes the number of defects found within a given testing or inspection budget. Through the early identification of potential defects in the software development lifecycle, software defect prediction (SDP) models play a proactive role in enhancing software quality and reliability [1,2]. Consequently, this contributes to the overall dependability and security of a software system, minimizing the occurrence of software failures and vulnerabilities.

Over the past decades, considerable advancements in SDP have been reported with the introduction of each new model, as supported by numerous publications [3–9]. However, recent studies have brought to light a disparity between the perceived advancements in SDP and the actual achievements [10–12]. Zhou et al., in their comparison of simple module size models and complex SDP models published from 2002 to 2017, reported comparable or superior prediction performance of the former over most existing models in the literature [12]. Similarly, in 2021, Zeng et al. demonstrated that a simple logistic regression model outperformed deep learning models that were touted as the state-of-the-art in SDP, in terms of both effectiveness and efficiency across multiple projects [11]. These findings reveal a substantial disparity between the reported results within the academic community and the true progress made in SDP over the past decades. Essentially, the performance of most (if not all) models appears to have been overestimated in their original evaluations.

The tendency to overestimate the effectiveness of defect prediction within our community arises from a fundamental issue: the absence of a consistent evaluation framework for newly proposed SDP models. Currently, researchers lack consensus when it comes to comparing these models with previous ones, including aspects such as baseline models, classification threshold settings, and performance indicators. This lack of consensus has several consequences. First, the absence of a global reference point impedes our ability to determine the true extent of progress being made in defect prediction [13]. Second, in comparative experiments, models often rely on their default classification thresholds [14,15], leading to misaligned measurements of SQA (Software Quality Assurance) efforts and potentially unfair comparisons. Third, the diverse range of performance indicators, coupled with insufficient reporting of prediction results, makes it impractical to compare studies comprehensively. Different performance indicators do not always align, resulting in situations where a model may demonstrate higher performance under one indicator but lower performance under another [16]. These challenges, arising from the lack of standardized evaluation frameworks and inconsistent reporting practices, contribute to the overestimation of defect prediction effectiveness within the field.

Undoubtedly, these problems pose a significant risk within our community, potentially leading to unintentionally misleading conclusions regarding the state-of-the-art in defect prediction. This, in turn, can misinform practitioners about the true predictive performance of newly proposed SDP models and which models are genuinely superior. Urgently, our community must identify truly superior models that can provide tangible benefits to SQA engineers in practical scenarios. To this end, conducting objective evaluations of the effectiveness of newly proposed SDP models is of utmost importance. Unfortunately, the importance of objective performance evaluation has not received the necessary attention it deserves within our community. Among the three problems concerning objective performance evaluation, only the baseline model has garnered some attention (albeit insufficient) [13,17,12], while the issues regarding appropriate threshold settings and performance indicators remain largely unexplored.

In this study, our objective is to introduce MATTER (a fraMework towArd a consisTenT pErformance compaRison) as a solution to address the aforementioned problems. MATTER comprises three essential components: establishing a global reference point for comparison, ensuring fair performance comparison, and achieving consistent performance evaluation. To this end, we first introduce ONE (glObal baseliNe modEl), a model with the 3S properties (Simplicity in implementation, Strong predictive ability, and Stable prediction performance), to serve as the baseline against which other models are evaluated. Second, we propose the SQA-effort-aligned threshold setting to ensure that performance comparisons are conducted in a fair manner, taking into account the effort required for software quality assurance. Third, we advocate for consistent performance evaluation by adopting a set of core performance indicators that reflect the practical value of prediction models in achieving real progress. By applying MATTER to the same benchmark datasets, researchers and practitioners can obtain more accurate and meaningful insights into the performance of defect prediction models. Ultimately, this framework will contribute to better gauging our progress in the field of defect prediction and enhancing the overall quality of software development.

Our main contributions can be summarized as follows:

- We conduct a thorough examination of the drawbacks in current methodologies to evaluate the effectiveness of newly proposed defect prediction models. We show that, for a new model, the existing evaluation approach may lead to a biased evaluation due to the lack of the ongoing use of a global “stable” baseline model, a fair performance computation setting, and the unified performance indicators. Consequently, the documented advancements in defect prediction found in literature may lack reliability or be misleading.
- We propose MATTER, an evaluation framework designed specifically for assessing newly proposed SDP models. In contrast to existing approaches, our framework emphasizes the continuous use of a simple, strong, and stable baseline model as the global reference point, an SQA-effort-aligned threshold setting for fair performance comparison, and three core practical cost-effectiveness indicators for consistent performance comparison. With MATTER, researchers can determine the practical relevance of a newly proposed SDP model for practitioners. Importantly, this framework enables us to achieve genuine progress in defect prediction, fostering a stable and healthy development within the defect prediction community.
- Using MATTER, we conducted an experiment to assess the real advancements in defect prediction. To our surprise, our findings indicate that none of the representative SDP models we investigated surpass the prediction performance of the simple baseline

model, ONE. These results not only demonstrate the effectiveness of our framework in accurately capturing genuine progress in defect prediction but also serve as a cautionary reminder that the anticipated advancements in defect prediction may not unfold as initially envisioned.
- We offer the source code for our study, enabling researchers to readily utilize MATTER for evaluating the efficacy of newly proposed models [18]. This availability will aid in the development of genuinely effective SDP models, allowing for further advancements in the field.

The structure of this paper is as follows. Section 2 provides an overview of the relevant background. In Section 3, we delve into a comprehensive analysis of the limitations of existing model evaluation practices. Section 4 introduces our proposed framework, MATTER, which ensures consistent model evaluation. The experimental setup is detailed in Section 5, while Section 6 presents the experimental results in detail. Section 7 discusses our findings, including the influence of important factors, implications for defect prediction, and threats to validity. In Section 8, we summarize the related work. Finally, Section 9 concludes the paper, outlining potential avenues for future research.

2. Background

In this section, we introduce the background, including the scenarios and models in defect prediction.

2.1. Defect prediction scenarios

Within real-world software development, it is common for SQA resources to be limited [19]. Consequently, it becomes impractical to inspect or test every module in a given project. This naturally gives rise to the question: which modules should be chosen for SQA? Defect prediction has emerged as a promising approach to addressing this problem. The process involves predicting the likelihood of defects in each module within the project, thereby enabling the selection of modules with a higher defect probability for SQA activities. In this context, the two primary SDP scenarios that have received significant research attention are classification and ranking.

In the classification scenario, modules within a target project are initially divided into two categories: “defective” and “clean,” based on a predetermined threshold. If a module has a predicted defect-proneness exceeding the threshold, it is categorized as “defective”; otherwise, it is classified as “clean.” This scenario operates under the implicit assumption that the available SQA resources are sufficient to inspect or test all the modules classified as “defective.” The goal is to achieve an effective classification that enables SQA engineers to identify as many defects as possible while maintaining a reasonable level of precision.

In the ranking scenario, the modules within a target project are ordered based on their predicted defect-proneness, with the highest predicted values at the top and the lowest at the bottom. This ranked list allows SQA engineers to select modules for quality assurance in a sequential manner, starting from the top and continuing until the available inspection or testing resources are fully utilized. Unlike the classification scenario, the ranking scenario does not assume any specific quantity of available resources. In practical terms, given a predetermined budget for testing or inspection efforts, an effective module ranking is expected to aid SQA engineers in identifying defects in the target project as early as possible and maximizing their discovery within the given resource constraints.

2.2. Defect prediction models

To build an SDP model, the prevailing approach involves utilizing software metrics that quantify code complexity [20], development processes [21,22], and even characteristics of the development organization [23] as predictors. Within the defect prediction community, two primary modeling techniques, namely supervised and unsupervised techniques, are widely employed for building SDP models.

In the case of supervised SDP models, the objective is to develop a function, either explicitly or implicitly, by leveraging labeled training data. This function aims to best capture the relationship between software metrics and observed defect-proneness within the data. The key advantage of supervised models is that they can utilize prior knowledge to establish connections between predictors and defects, enabling accurate predictions on unseen data. Over the past decades, numerous supervised models have been proposed, encompassing traditional regression-based [24], rule-based [25], tree-based [26], and modern deep-learning-based models [27]. However, the main drawback of supervised defect models lies in the potential challenge of obtaining or having access to labeled training data, which can be expensive or even unavailable in certain cases [28–30].

In contrast, unsupervised SDP models operate without the requirement of labeled training data. Within the literature, unsupervised models primarily encompass association-based methods [12] and clustering-based techniques [31,32]. Unlike supervised models, unsupervised SDP models typically have lower costs associated with model development since they do not rely on label information. This absence of label dependence allows for a more efficient building process.

3. Pitfalls in current model performance evaluation practices

When introducing a new defect prediction model, it is essential to evaluate its effectiveness. The evaluation procedure commonly followed in the literature involves several steps. First, a set of previous SDP models is selected as a basis for comparison with the

Table 1
The usage of baseline models in recent representative SDP studies within the latest six years (2017–2022). (See Appendix A [18] for the meaning of baseline model b_i, $1 \le i \le 33$).

2021		2022				2023		
KSETE	top-core	STABILIZER	MEG	EASC	FENSE	DSSDPP	TDTSR	ADA
[7]	[38]	[39]	[40]	[17]	[41]	[42]	[3]	[43]
b1∼b5						b1,b5	b4	b1
	b6∼b9							
b11∼b12		b12			b12		b10	
				b13∼b18				
							b19	
		b20						b20
			b21∼b26					
					b27,b28			
						b29,b30		
								b21∼b33

Table 2
The mean G1 values on the same five target projects (EQ, JDT, Lucene, Mylyn, and PDE) in AEEEM for the newly proposed defect prediction models as well as the used baseline models in different studies.

New model	G1	Baseline model						
		TCA+	CCA+	HDP-KS	B.Filter	TNB	VCB	Bellwether
KSETE [7]	0.653	0.283	0.278	0.024	0.339	0.406		0.478
MSMDA [6]	0.696	0.621	0.615	0.613	0.593			
Bellwether [13]	0.750	0.688				0.654	0.558	0.750
SSTCA+ISDA [4]	0.771	0.521					0.638	

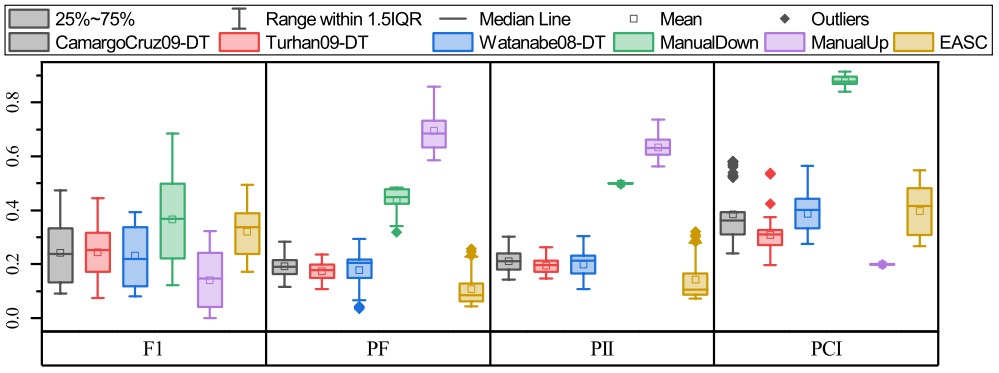
newly proposed model. Second, these models, including the newly proposed one, are applied to the same test set using the same training set. Performance values are calculated to assess the prediction capabilities of each model. A default value is often employed as the classification threshold, and commonly used performance indicators include recall, precision, F1 score, G1 score, AUC, and MCC [33,12]. Third, after obtaining the performance values, statistical methods such as the Wilcoxon signed-rank test [34] are employed to determine if the newly proposed model exhibits significant improvement over the baselines. Additionally, the effect size, such as Cliff's delta [35], may be used to estimate the magnitude of observed differences [36,17]. This model evaluation process has been applied to evaluate hundreds of SDP models in the existing literature [37].

In the subsequent sections, we shed light on three critical problems present in the aforementioned model evaluation process. These issues significantly impede our comprehension of the genuine progress achieved within the SDP community.

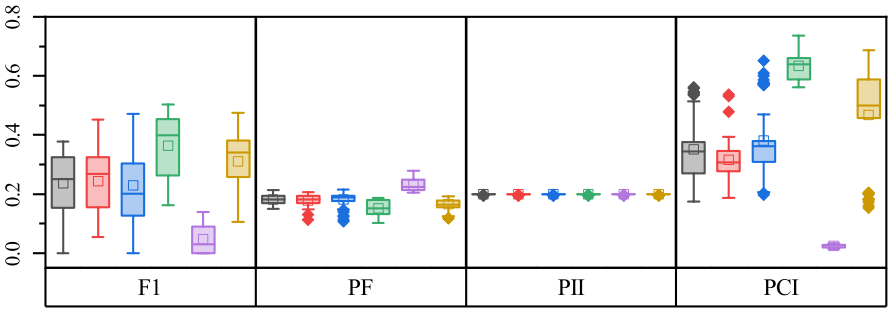
3.1. Absence of a common "stable" baseline model as the global reference point

In our manuscript, a baseline model refers to a simple, often naive, predictive model that serves as a reference point for comparison with more complex or sophisticated models. Common examples of baseline models include random guessing, ManualDown (descending sorting modules by LOC), ManualUp (ascending sorting modules by LOC), etc. In an ideal scenario, each time a new SDP model is introduced, it should be compared against a standardized baseline model. This practice is essential to accurately gauge the progress being made in defect prediction. To assess the prevalence of this practice, we conducted a search for representative defect prediction studies that meet the following criteria: they were published in renowned journals and conferences, were conducted within the past three years (2021-2023), and proposed novel defect prediction models, representing the state-of-the-art. After applying these criteria, we identified 9 defect prediction studies that satisfied the conditions [7,38–40,17,41,42,3,43]. Detailed literature selection process can be found in our Appendix A [18]. Table 1 presents a summary of these studies along with the corresponding baseline models used. Surprisingly, we observed that more than 30 different baseline models were employed across the 9 studies. *Notably, there was no single baseline model that was consistently adopted across the studies.* This finding reveals that the current practice in defect prediction does not involve the use of a common baseline model.

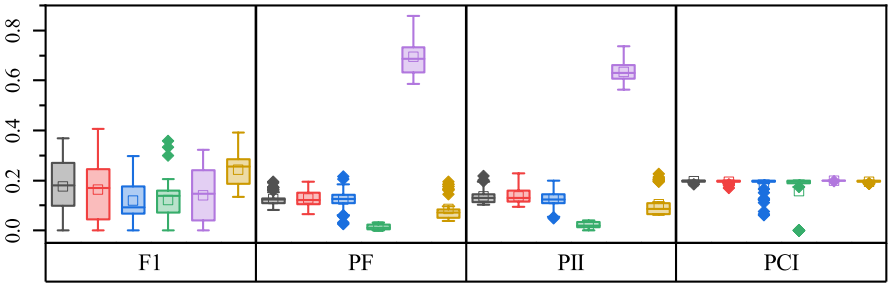
When utilizing a common baseline model, it is crucial to ensure consistent prediction performance on identical test data across different studies. Otherwise, such a baseline model cannot serve as a reliable global reference point for defect prediction. In Table 2, we present a summary of the mean G1 values for baseline models used in different studies, all of which employ the same five projects from AEEEM as the test data. *It becomes evident that the performance of the same baseline model exhibits substantial instability across studies when evaluated on the same test dataset.* This problem primarily arises from the characteristics of previous models, which are often supervised, complex, and not openly available as open-source implementations. Consequently, researchers are required to re-implement these models as baselines in subsequent studies. Due to implementation bias and varying training data, it is challenging for the same baseline model to achieve identical performance values on the same test data in different studies.



(a) SQA-effort-unaligned



(b) PII-aligned



(c) PCI-aligned

Fig. 1. The comparison of EASC with the baseline models on the MA-SZZ-2020 data set under the (a) SQA-effort-unaligned, (b) PII-aligned, and (c) PCI-aligned setting.

3.2. Unfair model comparison due to SQA-effort-unaligned threshold setting

In real-word development, the SQA effort is often constrained when inspecting or testing potentially defective modules identified by SDP models. Therefore, SQA engineers strive to maximize the detection of real defects within the limitations of their effort. According to the literature [36], the SQA effort involved in defect prediction primarily encompasses code inspection/testing and context-switch efforts. On one hand, larger modules tend to require more code inspection/testing effort. On the other hand, examining a greater number of modules necessitates additional context-switch effort. Taking these factors into account, it is crucial to align the SQA effort when comparing the effectiveness of different SDP models. Failing to do so may lead to misleading conclusions.

In current practice, it is common to use default threshold settings for conducting model performance comparisons. However, these default thresholds are often not aligned with the effort required for SQA. This means that the effort required to inspect or test predicted defect-prone modules may not be the same for different prediction models. For example, when EASC was proposed [17], it was compared with four supervised models (CamargoCruz09-DT [44], Menzies11-RF [45], Turhan09-DT [46], and Watanabe08-DT [47]) and two unsupervised models (ManualDown [12] and ManualUp [12]). All the supervised models (including EASC) used a default threshold of 0.5 to classify the modules in a project into “defective” or “clean”. Fig. 1a reports the comparison of EASC with the baseline models on a quality defect dataset called MA-SZZ-2020 [30] under the SQA-effort-unaligned threshold setting. The metrics used in the comparison were F1, PF (probability of false alarm), PII (Percent of Instances Inspected/tested [17]),

Table 3
The usage of performance indicators in recent representative SDP studies within the latest six years (2017–2022).

2021		2022				2023		
KSETE	top-core	STABILIZER	MEG	EASC	FENSE	DSSDPP	TDTSR	ADA
[7]	[38]	[39]	[40]	[17]	[41]	[42]	[3]	[43]
i2,i3		i1∼i3		i2,i4	i1,i2,i4	i3	i1∼i4	i4
i5∼i9		i9	i6	i7∼i9	i7,i8	i6,i7	i7	i5
	i10			i10				
		i11			i11			
				i12∼i16				
								i17

The $i1$ to $i17$ represent precision, recall, PF, F1, G1, MCC, AUC, recall@20%LOC, IFA, Popt, Popt20, PII@20%LOC, PII@1000LOC, PII@2000LOC, recall@1000LOC, recall@2000LOC, balanced accuracy, respectively.

and PCI (Percent of Code Inspected/tested). The results suggest that EASC demonstrates a competitive performance, particularly outperforming ManualDown.

However, it is important to note that the aforementioned conclusion is based on the SQA-effort-unaligned threshold settings, which may render the across-model performance comparison biased. To address this concern, we examined the comparison results with fixed parameters. Specifically, Fig. 1b presents the findings when the PII is set at 20%. Upon closer inspection, we observe that ManualDown exhibits a slightly higher F1 score, a slightly lower PF value, and a slightly higher PCI compared to EASC. Furthermore, Fig. 1c displays the comparison results when the PCI is fixed at 20%. Notably, despite ManualDown having a lower F1 score, both the PF and PII values are also lower than those of EASC. These results indicate that ManualDown remains competitive with EASC when considering the cost-effectiveness aspect, i.e., the number of defects detected per unit effort expended.

The above observations highlight the potential pitfalls of using SQA-effort-unaligned threshold settings for evaluating model performance, as it can inadvertently lead to misleading conclusions. This problem is prevalent in defect prediction studies and necessitates urgent attention within our research community. Failing to address this problem can result in misleading conclusions that may misguide both researchers and practitioners in their decision-making processes.

3.3. Unfeasible across-study comparison due to diverse performance indicators

To accurately assess the progress in defect prediction, it is essential to compare the predictive performance of models across different studies in a reliable manner. This comparison depends on the use of consistent test sets as a fundamental requirement. However, it is also crucial to ensure that identical prediction performance indicators are employed across various studies, even when utilizing the same test sets. This consistency facilitates meaningful assessments and allows for accurate evaluation of the advancements made in defect prediction.

Unfortunately, it is widespread in the literature where different studies adopt various performance indicators. In Table 3, we summarize the performance indicators utilized in the 9 representative defect prediction studies presented in Table 1. It is evident that these 9 studies employ a total of 17 different indicators. Notably, there is no single common performance indicator employed across all 9 studies, making it challenging to directly compare the performance of the proposed new defect prediction models, even when using the same test sets. Additionally, attempting to infer a common performance indicator from the diverse set of indicators used in different studies for an across-study model performance comparison proves to be difficult. This difficulty arises from the fact that these studies often lack sufficient information necessary to enable such inference.

The aforementioned practice creates obstacles in comparing models across different studies. Consequently, when a new model is presented as superior to previous models in a study, it becomes difficult to ascertain if it genuinely advances the state-of-the-art. To address this issue, it is necessary to utilize consistent and appropriate performance indicators across various studies. These indicators should effectively evaluate the practical effectiveness of models in real-world scenarios. However, this presents a challenge since different indicators serve different purposes, and there is no consensus on which indicators should be universally adopted.

4. MATTER: a fra**M**ework tow**A**rd a consis**T**en**T** p**E**rformance compa**R**ison

In this section, we propose MATTER (a fra**M**ework tow**A**rd a consis**T**en**T** p**E**rformance comparison), which aims to address the challenge of consistent performance comparison in defect prediction. MATTER comprises three key components: a common “stable” baseline model, effort-aligned threshold settings, and core indicators. First, we propose the use of ONE, an easy-to-use and stable baseline model, as “the global reference point” for evaluating the effectiveness of any newly proposed model m in defect prediction. ONE serves as a reliable benchmark against which the performance of model m can be evaluated. Second, we advocate for the adoption of two effort-aligned settings to ensure fair performance comparisons between model m and ONE. These settings are designed to align with SQA effort, enabling a more realistic evaluation of the models. Third, we propose the inclusion of three core performance indicators in all studies to facilitate a standardized evaluation. These indicators should be consistently reported, along with detailed prediction results on each test set. This approach enables feasible across-study comparisons and ensures a reliable evaluation of model performance.

Fig. 2 illustrates the evaluation workflow in MATTER. When a new SDP model m is proposed, the following steps are used to evaluate its effectiveness.

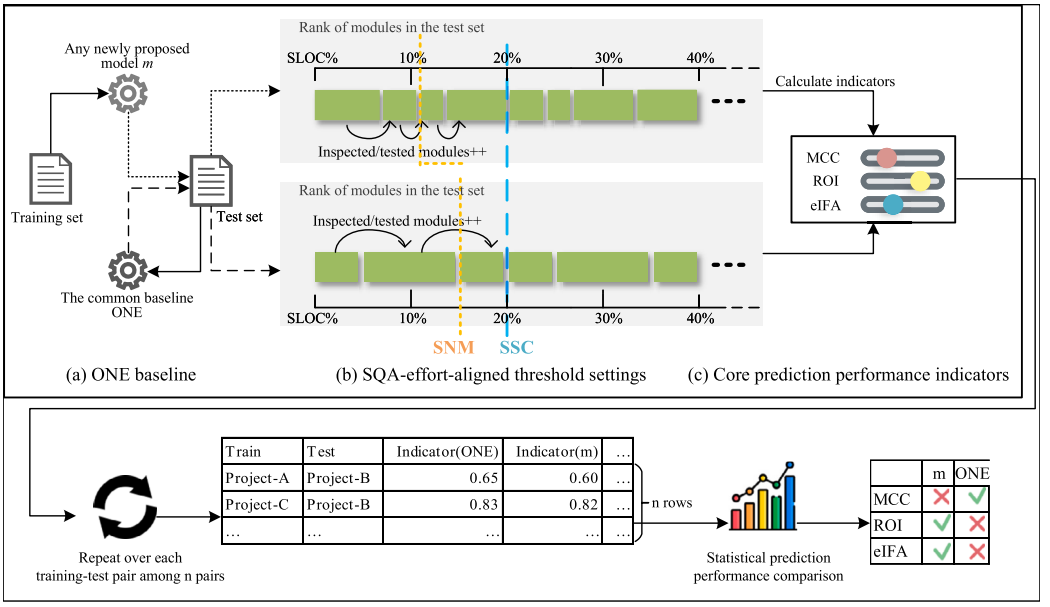


Fig. 2. MATTER: A framework for consistent comparisons between SDP models.

1. Generate defect-proneness rankings for *m* and ONE. On the same test set, for each model, rank the modules from the most to the least predicted defect-proneness. As a result, two rankings are obtained, one for model *m* and another for ONE;
2. Apply two SQA-effort-aligned threshold settings to obtain the corresponding classification results: SNM (Same Number of Modules) and SSC (Same Size of Code). The purpose is to enable a comparison under the same inspected/tested number of modules (SNM) or the same inspected/tested code size (SSC).
3. Calculate three core prediction performance indicators for each model: MCC, ROI (Return On Investment, i.e., the number of defective modules found per unit SQA-effort), and eIFA (SQA-effort required in Initial False Alarm).
4. Repeat (1)–(3) over each training-test pair to obtain the performance values. Consequently, if there are *n* training-test pairs, for each indicator, both *m* and ONE will have *n* prediction performance values.
5. Conduct a statistical prediction performance comparison. Under each indicator, use a statistical method to examine if *m* is significantly better than ONE. If there is a significant difference at the significance level of 0.05, examine whether the effect size (say Cliff's delta [35]) is non-trivial.
6. By consistently applying MATTER to the same benchmark test datasets in different studies, our community can achieve a uniform and consistent performance evaluation for SDP models. This enables the identification of practically effective prediction models and contributes to the advancement of state-of-the-art in defect prediction in a trustworthy manner.

4.1. Using a simple, strong, and stable baseline model as "the global reference point"

A solid baseline model should possess the following key attributes: (1) simple to implement without bias [48]; (2) strong in prediction ability, providing a benchmark "floor performance" to filter out weaker models [13]; and (3) stable in prediction performance across studies, serving as a reliable global reference point [26,13,48]. A model lacking the 3S properties (simple, strong, and stable) is unsuitable as a standardized global reference for evaluating model performance consistently.

In MATTER, ONE is specifically designed to adhere to the 3S properties. Algorithm 1 presents the pseudo-code for ONE. First, for a given target project, the modules are ranked in descending order based on their module size measured by SLOC (lines 2-3). Then, the resulting module rank list, denoted as *L*, is split into two parts: *E*, representing the top largest modules accounting for 20% of the total code size, and *R*, encompassing the remaining modules (lines 4-5). Following this, the modules in *E* are ranked in ascending order by size and appended to the end of *R* (lines 6-7). Finally, the performance value PM is calculated and returned (lines 8-9).

As can be seen, ONE prioritizes modules with moderate size over small or very large modules for inspection and testing. This prioritization is based on the following relationships between module size and defect-proneness [12]: (1) larger modules are more likely to contain defects; (2) very large modules may exhibit a lower defect density; and (3) smaller modules tend to have higher defect densities. Consequently, ONE avoids falling into common traps, such as (1) the need for high context switching effort when inspecting/testing numerous small modules and (2) the high code inspection/testing effort required for very large modules. Therefore, intuitively, ONE offers a cost-effective defect prediction. By following the module ranking provided by ONE for inspection/testing, defects can be identified reasonably quickly, and a reasonable number of defects can be found per unit SQA effort.

In essence, ONE is an unsupervised model that relies solely on the code size of modules in a target project, eliminating the need for training data. This characteristic leads to low data collection and model building costs. As a result, ONE is straightforward to implement without introducing implementation bias. Additionally, as highlighted in Section 6, ONE exhibits competitiveness or even

Table 4
Difference between ONE and previous SDP baseline models.

baseline	Stable (unsupervised)	Simple (only based on loc metric)	Strong(one model for both effort-aware and non-effort-aware)
Bellwether [13]	NO	NO	YES
EASC_E & EASC_NE [17]	NO	NO	NO
SC [32]	YES	NO	YES
CLA [31]	YES	NO	YES
FCM [49]	YES	NO	YES
ManualDown & ManualUp [12]	YES	YES	NO
ONE	YES	YES	YES

superiority when compared to representative defect prediction models. This strength positions ONE as a valuable tool for selecting practical and useful prediction models. Notably, ONE maintains consistent (i.e., stable) prediction performance across different studies, regardless of variations in the training sets. This stability allows ONE to serve as a global reference point, facilitating cross-study comparisons of model performance. Consequently, the use of ONE as a baseline model fulfills all three of the 3S properties.

In contrast, the previously proposed baseline models reviewed in our study did not fully meet the requirements of the 3S. As can be seen in Table 4, in terms of "stable", Bellwether and EASC are supervised models that cannot perform stably on a test set since their performances are influenced by training data; moreover, in terms of "simple", all of them are not as simple as ONE, except for ManualDown and ManualUp, which are only based on the code size of modules; at last, in terms of "strong", while ManualDown, ManualUp, EASC_E, and EASC_NE are designed for high performance on a single aspect (effort-aware or non-effort-aware), which means the other aspect will be sacrificed. In conclusion, compared to those previous baseline models, ONE emerges as a robust baseline model that fulfills all three of the 3S properties, offering simplicity, stability, and strength.

Algorithm 1 ONE's pseudocode.

```
procedure ONE(targetProject)
    S = getModuleSet(targetProject)
    L = rankModuleInDscendingOrderBySize(S)
    E = getTopLargestModules(L)
    R = L − E
    E = rankModuleInAscendingOrderBySize(E)
    N = catenate(R, E)
    PM = calculatePerformance(N)
    return PM
end procedure
```

4.2. Using the SQA-effort-aligned threshold setting to make a fair performance comparison

Given two models, in real-world development, it is natural to ask the following two questions: what if an SQA engineer inspects/tests the same number of modules according to their recommendations? what if an SQA engineer inspects/tests the modules with the same code size according to their recommendations? In nature, the former aims to conduct a model comparison under the alignment of context switching effort, while the latter aims to conduct a model comparison under the alignment of code inspection/testing effort.

When ONE is used as the baseline model, how to evaluate the effectiveness of a new model *m* in comparison to ONE? In order to keep consistent with the current practice, we recommend using the following SQA-effort-aligned threshold settings to make a fair comparison between *m* and ONE:

- SNM (the Same Number of inspected/tested Modules). Select an equal number of top-ranked modules from the defect-proneness ranking lists provided by *m* and ONE for inspection and testing to align their context switching efforts.
- SSC (the Same Size of inspected/tested Code). Select an equal code size of top-ranked modules from the defect-proneness ranking lists provided by *m* and ONE for inspection and testing to align their code inspection/testing efforts.

By consistently applying the above settings across different studies, we ensure fair model comparisons within a study and enable fair comparisons across different studies. This approach allows us to use ONE as a universal reference point to determine genuine advancements in defect prediction.

4.3. Using three core indicators as a starting point for consistent performance evaluation

From a practical standpoint, for a new defect prediction *m*, practitioners are particularly interested in assessing two key aspects compared with the baseline model. First, whether will *m* lead to a higher ROI (Return Over Invest, i.e., the percentage of defects found per unit SQA-effort)? A higher ROI means that more defects would be detected within the same SQA-effort by utilizing model *m*. Second, will *m* reduce the SQA effort on non-defective modules before identifying the first actual defective module? If *m* requires a lower SQA-effort for Initial False Alarms (IFA), the first actual defective module will be found more quickly.

Considering the above factors, ROI and eIFA (SQA-effort required in IFA) emerge as the two fundamental indicators for consistently evaluating and comparing the performance of SDP models. These indicators provide valuable insights into the model's ability to improve fault detection efficiency and optimize resource allocation during the software quality assurance process.

For a target (test) project p containing k modules, assume that:

- n_i: the actual defect label of the i-th module (0 for non-defective and 1 for defective).
- s_i: the code size (in SLOC) of the i-th module.
- x: the number of the top-ranked modules to be inspected/tested recommended by model m.

Consequently, the total number of actual defective modules in project p is:

$$n = n_1 + n_2 + \cdots + n_k \tag{1}$$

and the total code size in SLOC in project p is:

$$s = s_1 + s_2 + \cdots + s_k \tag{2}$$

Let TP be the total number of actual defective modules in top x ranked modules, PII be x/k, and PCI be the ratio of code size in top x ranked modules to the total code size in the project p.

ROI (**R**eturn **O**n **I**nvestment). We follow prior studies [50,17] to use PCI as the proxy of code inspection/testing effort and use PII as the proxy of context-switch effort. As mentioned in Section 4.2, model comparison is conducted under the following SQA-effort-aligned settings: SNM and SSC. For an SDP model, under SNM, where context switching efforts are aligned, ROI is defined as:

$$ROI = \frac{TP}{PCI} \tag{3}$$

Under SSC, where code inspection/testing efforts are aligned, ROI is defined as:

$$ROI = \frac{TP}{PII} \tag{4}$$

As observed, ROI denotes the number of actual defective modules found per unit code inspection/testing effort under SNM and the number of actual defective modules found per unit context-switch effort under SSC. By comparing the ROI values, we can assess whether the new SDP model *m* is more cost-effective in finding defects than the baseline model ONE.

eIFA (SQA-**e**ffort required in **I**nitial **F**alse **A**larms). For an SDP model, assume that y is the number of top-ranked modules to be inspected/tested before encountering the first true defective module in the defect-proneness ranking (in descending order). Consequently, the code inspection/testing effort and the context-switch effort associated with initial false alarms are respectively:

$$PCI_{\text{IFA}} = \frac{1}{s}\sum_{i=1}^{y} s_i \tag{5}$$

$$PII_{\text{IFA}} = \frac{y}{k} \tag{6}$$

As a result, the eIFA of model m is be defined as:

$$eIFA = \alpha \times PII_{\text{IFA}} + (1-\alpha) \times PCI_{\text{IFA}} \tag{7}$$

By default, we set a = 0.5.

MCC (**M**atthews **C**orrelation **C**oefficient). In addition to ROI and eIFA, we recommend MCC as the third essential indicator, which measures the correlation between the actual and predicted binary classifications.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{8}$$

By utilizing MCC, we can determine whether the prediction is better than a random model. An MCC value greater than 0 indicates that the prediction performs better than a random model [51].

Relationships with other indicators. In the literature, there are various performance indicators. Why do ROI, eIFA, and MCC matter?

- ROI. In [17], recall@20%SLOC and PII@20%SLOC serve as effort-aware indicators. Notably, 20%SLOC represents a specific instance of SSC, while ROI can be considered as their amalgamation for model comparison purposes. A more detailed discussion of the nature of ROI can be found in Appendix E [18]
- eIFA. In [17,52,53], IFA is utilized to quantify the modules inspected or tested until the discovery of the initial true defective module. eIFA is an upgraded variant of IFA that encompasses the effort expended on code inspection and testing.
- MCC. While F1 and AUC have been widely used as indicators in the literature, recent studies reveal that they can be unreliable and even misleading for model comparison [54,55]. In contrast, MCC is highly recommended as a more reliable alternative [51,56–58].

By incorporating ROI, eIFA, and MCC, developers can gain a holistic understanding of the effectiveness of a prediction model, facilitating informed choices.

In summary, we propose employing ROI, eIFA, and MCC as the fundamental indicators in MATTER for consistent model performance comparison. Additional indicators can be incorporated as necessary for specific situations. The crucial aspect is to maintain the consistent use of appropriate performance indicators when evaluating and comparing model performance within and across studies.

5. Experimental setup

In this section, we describe the research questions, compared defect prediction models, datasets, and evaluation strategies.

5.1. Research questions

Our study addresses three research questions (RQs). The first two RQs pertain to evaluating MATTER itself, while the final RQ investigates the current state of defect prediction using MATTER.

RQ1: Does aligning the SQA-effort in MATTER for comparing model prediction performance prove necessary? We perform an experiment to examine the performance ranking of various defect prediction models. If there is a significant shift in the ranking before and after aligning the SQA-effort, it suggests that the unaligned threshold setting distorts the conclusion of the model comparison in terms of prediction performance (in terms of fair comparison). The results will aid in determining the necessity of SQA-aligned threshold setting.

RQ2: Is ONE a strong baseline defect prediction model employed in MATTER? In the literature, various supervised models, such as Bellwether [13] and EASC [17], have been explicitly recommended as baseline models during their development. Additionally, several unsupervised models like SC [32] and CLA [31] that are relatively simple and independent of training data have the potential to serve as baseline models. It is crucial to determine whether ONE exhibits sufficient strength in defect prediction compared to these recommended or potential baseline models. If the answer is affirmative, it establishes ONE as a suitable baseline model in MATTER, providing a global reference point for evaluating newly proposed software defect prediction models.

RQ3: To what extent has progress been made in defect prediction according to MATTER? It is evident that existing SDP models aim to enhance the effectiveness of defect prediction and have reported progress in this aspect. However, a fundamental question arises: do representative defect prediction models outperform ONE within MATTER? Specifically, do they meet our expectations in terms of performance under MATTER? This RQ highlights the importance of evaluating defect prediction models within a consistent framework to truly gauge their practical effectiveness. The experimental results will provide insights into whether significant progress has been made in defect prediction based on the existing studies.

5.2. Compared defect prediction models

For the first two RQs, we use the same set of eight SDP models for comparison. The first set of five models has been acknowledged as baseline models in existing literature. The remaining three models have the potential to serve as baseline models, given their simplicity, independence from training data, and demonstrated performance in prior comparative experiments. Moreover, all eight models are either readily replicable or available as open-source implementations.

- **Bellwether**. Krishna et al. discovered that, in a community, there was an exemplary project (called bellwether) whose data yields the best predictions on all others [13]. Based on this finding, they suggested leveraging the bellwether project data to train a defect prediction model for new projects. We employ their replication package to construct the Bellwether model.
- **EASC_E and EASC_NE**. Ni et al. suggested employing EASC_E as the baseline model for effort-aware performance evaluation and EASC_NE as the baseline model for non-effort-aware performance evaluation [17]. We meticulously adhere to the process outlined in [17] to construct these two models.
- **SC**. Zhang et al. introduced the utilization of spectral clustering (SC) for defect prediction [32]. We leverage their replication package to construct the SC model.
- **CLA**. Nam et al. suggested employing Cluster Labeling Algorithm (CLA) to assign labels to an unlabeled dataset based on the magnitude of metric values [31]. We utilize their replication package to construct the CLA model.

Table 5
Setup of replicating compared defect prediction models.

Model	Have directly runnable replication kit	Our re-implementation approach
Bellwether	YES	-
EASC_E & EASC_NE	NO	EASC_E and EASC_NE use Naïve Bayse for classification, then resort instances based on LOC (lines of code) of modules. We use the sklearn implementation of Naïve Bayse.
SC	YES	-
CLA	YES	-
FCM	NO	FCM uses Fuzzy C-Means algorithm for classification. We use its sklearn implementation.
ManualDown, ManualUp	YES	-
CamargoCruz09-NB, Amasaki15-NB,Peters15-NB	YES	-
Top-core	YES	-
MSMDA	YES	-
KSETE	YES	-

- **FCM**. Li et al. discovered that Fuzzy C-Means (FCM) emerged as one of the top-performing unsupervised techniques [49]. Our study rigorously adheres to the process outlined in [49] to construct the FCM model.
- **ManualDown and ManualUp**. These two size-based unsupervised baseline models have been recommended in previous studies [12].

For the third RQ, we evaluate the following six representative defect prediction models:

- **CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB**. Herbold et al. conducted a comparative analysis of 114 SDP models on 86 defect datasets and reported these three models as the best performers [59]. We utilize their provided replication package to construct these models.
- **Top-core**. Qu et al. proposed the use of "coreness," a measure of module importance in the class dependency network, to enhance defect prediction effectiveness [38]. Their study demonstrated that top-core significantly improved the performance of defect prediction models when compared to various other SDP models. We employ their replication package to build the Top-core model.
- **MSMDA**. Li et al. introduced the multi-source selection based manifold discriminant alignment (MSMDA) approach for effectively utilizing multiple sources of training data [6]. Their study demonstrated that MSMDA outperformed state-of-the-art SDP models in terms of prediction performance. We utilize their replication package to build the MSMDA model.
- **KSETE**. Tong et al. [7] proposed a kernel spectral embedding transfer ensemble (KSETE) model for defect prediction. According to their results, KSETE outperformed the previous state-of-the-art SDP models significantly. We utilize their replication package to build the KSETE model.

We summarize the setup of replicating the above models in Table 5. As can be seen, among those compared models, only EASC_E/EASC_NE and FCM do not have a publicly available replication kit. We have contacted the author of EASC_E/EASC_NE to confirm the implementation details of their models to strictly follow the process described in Ni et al.'s paper [17] to build the EASC_E and EASC_NE models. FCM is a clustering algorithm in which each data point belongs to each cluster center on the basis of the distance between the center and itself to a certain degree. In our study, we strictly follow the process described in [49] to build the FCM model. Moreover, we simply use the sklearn implementation of FCM algorithm, which can avoid implementation bias. For the rest models which have publicly available replication kit, we run their code on the original dataset they provide to compare the prediction performance we got with the performance reported in their original manuscript. The performance difference between performance when actually execute their replication kits and performance reported in their manuscripts are acceptable. Therefore, for all compared models, our replications are solid.

Note that for the top-core model, the predicted defectiveness of a module relies on the "coreness" computed from a graph extracted from the source code of a target project. However, Qu et al. [38] provided such information for only eight test projects, and for the other test projects used in our study, this information is unavailable. Additionally, MSMDA has a high memory requirement, and our experiments reveal that it does not scale well to a large number of source projects, leading to "out-of-memory failure" issues. Taking these factors into consideration, we present the experimental results from top-core and MSMDA in Appendix B [18], separate from the main body of this paper. These results do not conflict with the results obtained from other models.

5.3. Datasets

Table 6 shows the usage of the above datasets in the literature we reproduce in this work. Among those datasets, AEEEM, JURECZKO, NASA, and ReLink are the most frequently used datasets (used more than five times among ten reproduced studies). However, according to [60], the NASA dataset suffers from serious label noise issues and may influence the evaluation of SDP models, therefore, we exclude it from our selection of datasets. As a result, we include the other three most frequently used datasets,

Table 6
Datasets used in the literature reproduced in this work.

Dataset	Literature	#Lit
AEEEM	Bellwether [13], EASC [17], SC [32], FCM [49], ManualDown/ManualUp [12], CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB [59], MSMDA [6], KSETE [7]	8
JURECZKO	Bellwether [13], EASC [17], SC [32], FCM [49], ManualDown/ManualUp [12], CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB [59], MSMDA [6], KSETE [7]	8
NASA	EASC [17], SC [32], FCM [49], ManualDown/ManualUp [12], CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB [59], MSMDA [6]	6
ReLink	Bellwether [13], EASC [17], CLA [31], ManualDown/ManualUp [12], CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB [59], MSMDA [6]	6
NETGENE	CLA [31], CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB [59]	2
Softlab	ManualDown/ManualUp [12], MSMDA [6]	2
Eclipse	ManualDown/ManualUp [12]	1
ISM	FCM [49]	1
JIRA	KSETE [7]	1
Top-core dataset	Top-core [38]	1

Table 7
An overview of the studied datasets.

Dataset	#Proj	#Rel	#Inst	%Defective	Metric type	Source
AEEEM	5	5	324∼1862	9%∼40%	product, process	[61]
JURECZKO	31	62	10∼965	2%∼99%	product	[62]
ReLink	3	3	56∼399	30%∼51%	product	[63]
MA-SZZ-2020	5	50	129∼1267	3%∼38%	product	[30]
IND-JLMIV+R	23	59	103∼1415	5%∼20%	process	[64]

AEEEM, JURECZKO, and ReLink in our experiments to stay consistent with the studies we reproduce. More than that, AEEEM, JURECZKO, and ReLink are classic defect datasets widely utilized in SDP research, providing a solid foundation for comparative analysis and benchmarking. Additionally, we include MA-SZZ-2020 and IND-JLMIV+R datasets, both newly published in 2020, to offer an opportunity to explore novel aspects of defect prediction and extend the scope of research. By incorporating these diverse datasets, our study aims to provide a comprehensive and up-to-date evaluation of defect prediction models, addressing both established benchmarks and emerging datasets in the field. In total, our dataset contains 179 versions of 67 projects.

Table 7 summarizes the statistical information of the datasets used in our study. The second and third columns report the number of projects and the total number of releases a dataset contains. The fourth and fifth columns report the range of number of instances and the range of percentage of defective modules. The last two columns report the types of metrics and the source of the dataset, respectively.

5.4. Evaluation strategy

Data preprocessing and feature selection. For compared models proposed from previous studies, we let them conduct data preprocessing as their original settings. This is because our goal is to compare the prediction performance of SDP models, therefore, we do not unify their own strategies for data preprocessing and feature selection. For our baseline model, ONE, we do not conduct data preprocessing since (1) data cleaning is unnecessary for ONE: ONE only depends on LOC (lines of code) to predict the defectiveness of modules, and the public datasets we use do not contain instances with missing LOC values, and (2) ONE does not include data re-balance strategy. And no special feature selection is conducted in ONE since the only feature it needs is LOC.

Training-test data split. We adopt the all-to-one strict CPDP setting to split the training and test sets. Specifically, for each release in a project serving as the test set, we combine all releases from external projects within the same dataset as the training set. However, for the KSETE model, which relies on matrix computations and does not scale well to large training sets, we apply a one-to-one CPDP setting. In this case, each release in a project serves as the test set, and we utilize each release from external projects within the dataset as a separate training set. For KSETE, we report the mean performance across all training sets for each test set. All models in our experiments are compared on the same test sets.

Performance indicators. In our experiments, we utilize three core performance indicators from MATTER (MCC, ROI, and eIFA). It is important to note that higher values of MCC and ROI indicate better prediction models, while lower values of eIFA signify better performance.

Statistical comparison. We use the non-parametric Scott-Knott ESD (NP SK-ESD) test [7,65,66] to divide SDP models into different groups. We follow [66] to first rank models in terms of prediction performance on each test set to create a ranking list (the purpose is to control for dataset-specific model performance, as some datasets may exhibit varying levels of susceptibility to bias) and then use the ranking list as the input to apply the NP SK-ESD test. The NP SK-ESD test produces groups of SDP models, ensuring that the ranking differences within the same group are negligible, while the differences between groups are significant. To

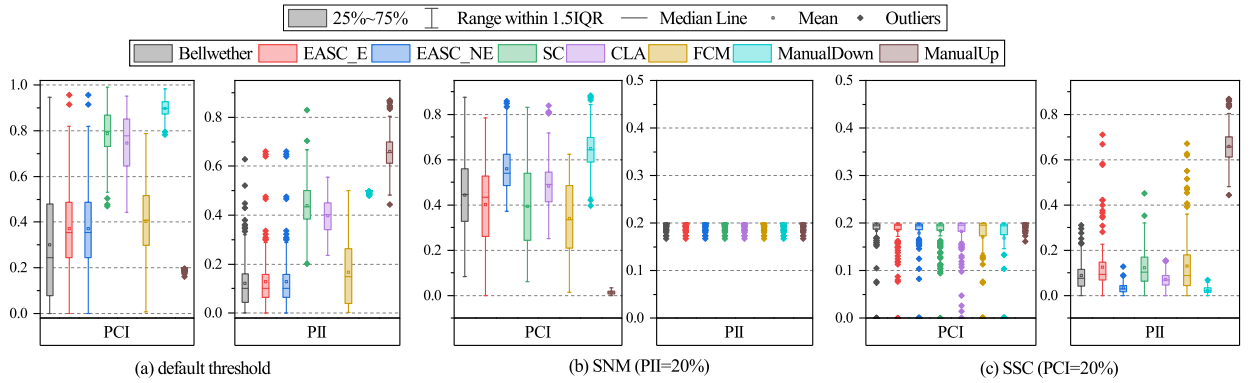


Fig. 3. The distribution of the required SQA-effort over the test sets for eight investigated models under (a) default thresholds, (b) SNM (PII = 20%), and (c) SSC (PCI = 20%).

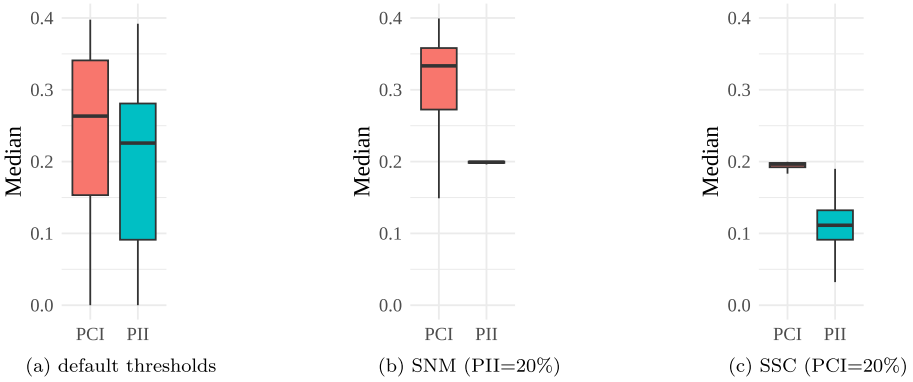


Fig. 4. The distribution of the median SQA-effort over eight investigated models under: (a) default thresholds, (b) SNM (PII = 20%), and (c) SSC (PCI = 20%).

determine the magnitude of the ranking differences between the compared models, We employ the Cliff δ effect size [67]: negligible for $|\delta| < 0.147$, small for $0.147 \leq |\delta| < 0.33$, medium for $0.33 \leq |\delta| < 0.474$, and large for $|\delta| \geq 0.474$ [68].

6. Experimental results

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

6.1. RQ1: Does aligning the SQA-effort in MATTER for comparing model prediction performance prove necessary?

Fig. 3 illustrates the distributions of the required code inspection effort (in terms of PCI) and context switch effort (in terms of PII) over the test sets for eight SDP models under: (a) default thresholds for individual models, (b) SNM (PII = 20%), and (c) SSC (PCI = 20%). Note that, under SNM (PII = 20%), the PII on some test sets may not exactly reach 20% due to the fact that 20% multiplied by the total number of modules may not result in an integer value. However, the PII values are as close to 20% as possible without exceeding it. Similarly, in practical scenarios, when applying SSC (PCI = 20%) with an SDP model at the module granularity, the minimum unit for selecting the code to be inspected is one module, rather than one line of code. Consequently, the PCI on some test sets may not precisely achieve 20%, but it is as close to 20% as possible without surpassing it. In order to get a clearer picture of the difference in SQA-effort under unaligned and aligned thresholds, Fig. 4 uses boxplots to summarize the distribution of the median SQA-effort for the eight investigated models.

Fig. 3(a) highlights the substantial disparity in code inspection effort (PCI) and context switch effort (PII) among SDP models under individual default thresholds. The median PCI ranges from approximately 20% to 90%, while the median PII ranges from about 10% to 65%. However, Fig. 3(b) and 3(c) demonstrate a significant reduction in SQA-effort variance with SNM and SSC. This finding is further reinforced by Fig. 4. The main takeaway from Fig. 3 and 4 is that the absence of threshold alignment results in a substantial SQA-effort variance across different SDP models, making fair performance comparisons impossible. It becomes unclear whether improved performance should be attributed to the SDP model's prediction power or the impact of unaligned SQA-effort. Aligning the thresholds enables relatively comparable SQA-effort among different SDP models, promoting fair performance comparisons.

When comparing different SDP models, the alignment of thresholds plays a crucial role in determining their performance ranking. Let us take MCC and PF as examples to demonstrate how unaligned thresholds can affect the performance ranking of different

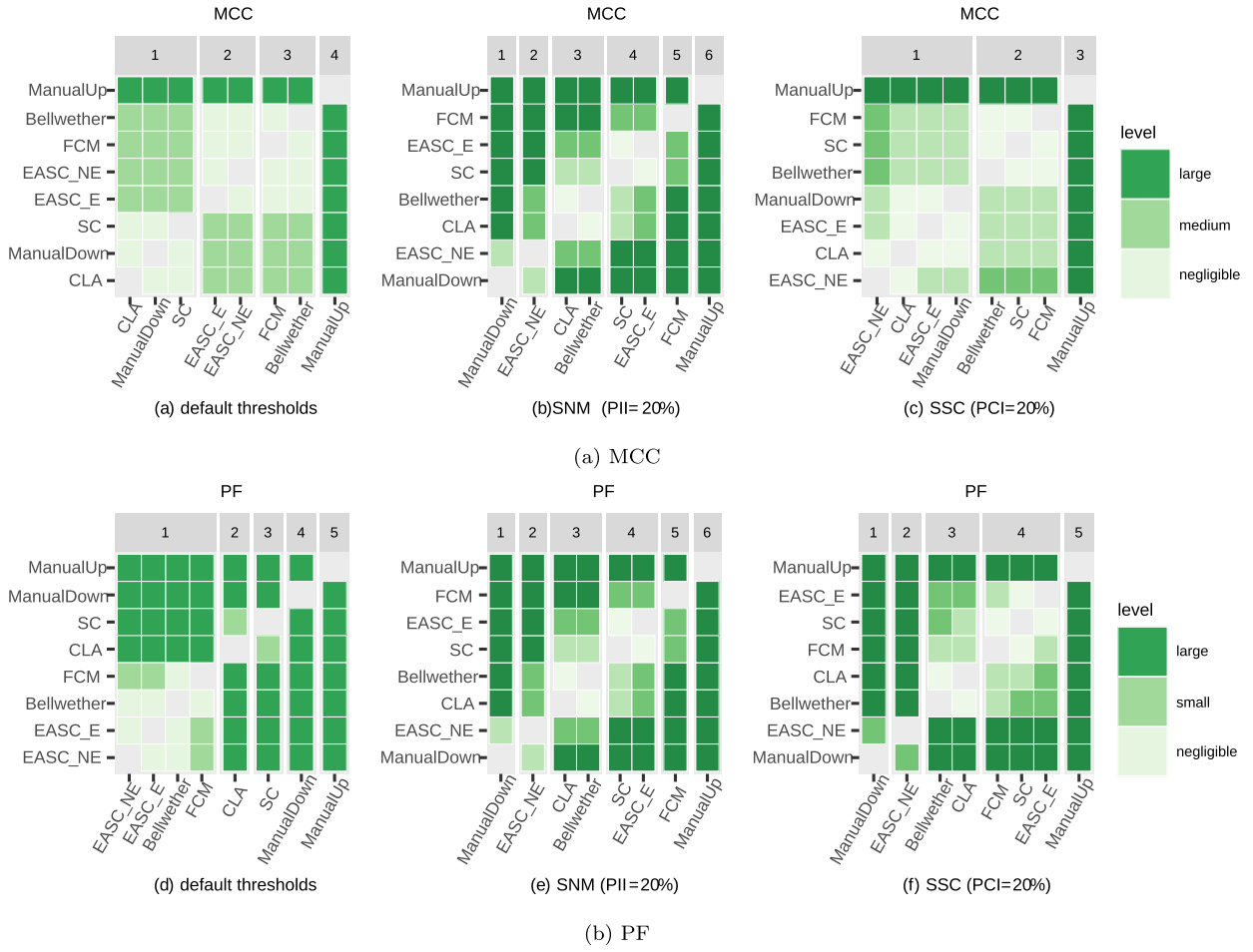


Fig. 5. Heatmap of the magnitude of the ranking difference of every two paired models of MCC and PF. The darker the color, the larger the magnitude of the ranking difference. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

models (similar observations can be made for other performance indicators). Fig. 5 illustrates the heatmaps showing the magnitude of ranking differences between paired models, along with the grouping results based on the NP SK-ESD test under three scenarios: default thresholds, SNM with PII = 20%, and SSC with PCI = 20%. It is evident that the choice of threshold alignment can lead to dynamic changes in the ranking of an SDP model.

- *An SDP model may have a top rank under the default threshold setting but have a low rank under the threshold-alignment setting.* Consider SC as an example. Based on the MCC analysis using the NP SK-ESD test, SC achieves the top rank of 1 when the default threshold setting (i.e., threshold-unalignment setting) is applied. However, its rank drops to 4 under SNM and drops to 2 under the SSC. By examining Fig. 3(a), it becomes evident that SC exhibits significantly higher values of PCI and PII under the default threshold setting compared to the other models. This suggests that SC recommends a considerably larger number of code and modules to be inspected. However, after aligning the thresholds, SC suggests a comparable number of code and modules to be inspected as the other models. This indicates that, under the default threshold setting, the effort required by SC in SQA heavily influences its high MCC rank. It is important to note that if we fail to consider the influence of SQA effort when comparing SC with other models, SC may obtain an artificially high MCC rank, falsely exaggerating its actual effectiveness in defect prediction.
- *An SDP model may have a low rank under the default threshold setting but have a top rank under the threshold-alignment setting.* Let us consider the example of ManualDown. According to the NP SK-ESD test, ManualDown obtains a rank of 4 (the second to last rank) in terms of PF under the default threshold setting. In [17], Ni et al. reported that ManualDown was much inferior to EASC_NE in terms of PF. However, regardless of whether the SNM or SSC setting is considered, its rank improves to 1 (the best rank). Upon examining Fig. 3(a), it is apparent that ManualDown suggests a significantly higher number of modules to be inspected, leading to an inflated PF. This setting creates an unfair comparison for ManualDown. However, as shown in Fig. 3(b) and Fig. 3(c), after aligning the threshold, ManualDown suggests a comparable number of modules and a similar amount of code to be inspected as the other models. The threshold-alignment setting causes a dynamic change in the FP rank of ManualDown, with it achieving the best rank. Interestingly, the performance rankings under the default threshold setting and SNM have a

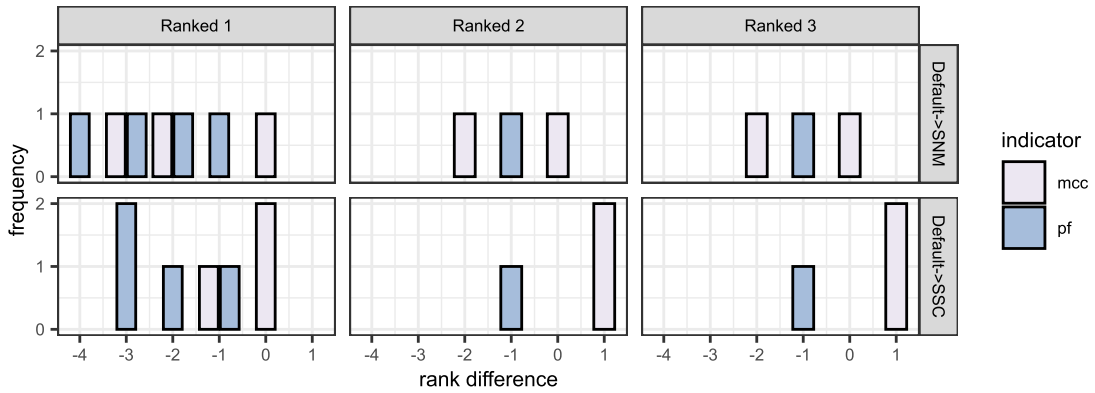


Fig. 6. The difference in the ranks for the metrics according to the SK ESD test grouping result. The bars indicate the frequency of the rank difference between the comparison under default thresholds and the comparison under SQA-aligned thresholds. In each column, models are grouped in Rank 1, Rank2, or Rank 3 in the comparison under default threshold. For each row, we investigate how ranks change from the default threshold setting to the SQA-effort threshold aligned settings (Default→SNM or Default→SSC).

Spearman correlation of only 0.116, while the rankings under the default threshold setting and SSC have a Spearman correlation of only 0.163. This indicates that if we do not control for the influence of SQA-effort when comparing ManualDown with the other models, ManualDown may have a falsely low PF rank that obscures its actual effectiveness in defect prediction.

In order to evaluate the impact of the threshold-alignment setting on performance rankings, we calculate the rank differences for models that appear in the top three ranks under the threshold-unalignment setting. For instance, if model m achieves a rank of 1 under the threshold-unalignment setting and a rank of 4 under the threshold-alignment setting, the rank difference of model m would be 1 - 4 = -3. Fig. 6 presents the rank differences for the models based on their prediction performance. It is evident that the models within the top three ranks exhibit significant instability. This suggests that aligning the thresholds can exert a significant influence on the resulting rankings of various SDP models. Failing to consider the alignment of SQA effort when comparing prediction performance may distort the rankings of different SDP models, potentially leading to misleading conclusions.

Answer for RQ1: *The use of unaligned threshold settings introduces significant variation in the required SQA effort among SDP models, which can strongly confound performance evaluations. By employing the threshold-alignment setting, the confounding effect can be mitigated, resulting in a fairer comparison of performance. Therefore, it is essential to utilize SNM and SSC to align the SQA effort in MATTER when evaluating prediction performance. This is particularly important considering the significance and relevance of SNM and SSC as two meaningful scenarios in defect prediction.*

6.2. RQ2: Is ONE a strong baseline defect prediction model employed in MATTER?

Fig. 7 presents the distribution of performance rankings for eight baseline SDP models and ONE. Notably, eIFA remains consistent under SNM and SSC, as it is unaffected by the SQA-effort threshold. Median and mean values of three core indicators under SNM (PII = 20%) and SSC (PCI = 20%) are summarized in Table 8. The comparison results for individual groups of test sets, including AEEEM, JURECZKO, ReLink, MA-SZZ-2020, and IND-JLMIV+R, are available in the online Appendix C [18]. These results support a similar conclusion to the overall dataset analysis.

Based on Fig. 7 and Table 8, the following observations can be made:

- *Of the nine compared models, ONE consistently performs at the top or upper-middle level under both SNM and SSC.* Under SNM, ONE achieves top-level performance (Rank 1) in terms of ROI and eIFA, and upper-middle-level performance (Rank 3 out of 7 groups) in terms of MCC. Under SSC, ONE achieves top-level performance in terms of MCC and eIFA, and second-level performance in terms of ROI.
- *ManualDown and ManualUp perform worse than ONE under both SNM and SSC.* While ManualDown performs well under SSC, it performs the worst in terms of ROI under SNM due to the significant code inspection effort required for the top largest modules. ManualUp performs as well as ONE in terms of ROI under SNM, but it performs the worst among the compared models in all other situations presented in Fig. 7 and Table 8.
- *EASC_E, CLA, Bellwether, SC, and FCM either perform worse or show similar performance compared to ONE under both SNM and SSC.* None of them outperforms ONE on any indicator under either SNM or SSC, indicating that they are not comparable to ONE in terms of prediction performance.

EASC_NE demonstrates similar performance to ONE in most cases. Under SNM, ONE achieves a median MCC of 0.219, slightly worse than the median MCC of 0.251 for EASC_NE. In terms of MCC under SSC (PCI = 20%), ONE achieves a median value of 0.145,

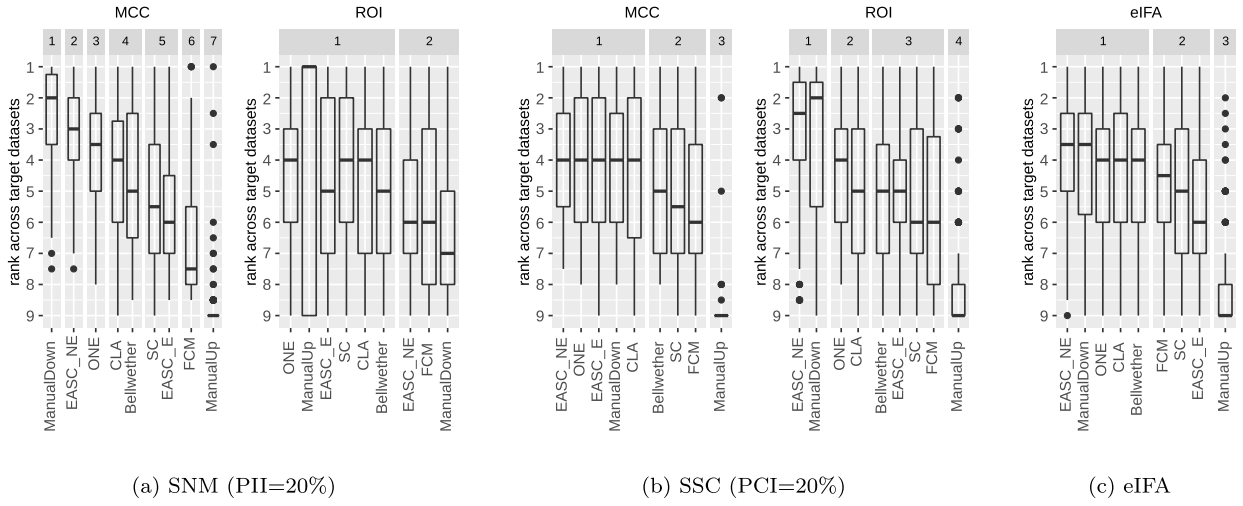


Fig. 7. The performance ranking distributions of different models by the NP SK-ESD test (The smaller the group number on the top of the plot, the better the performance ranking is).

Table 8
The median value and mean (standard deviation) value of representative models and ONE under MATTER.

	Median				
	MCC (SNM)	ROI (SNM)	MCC (SSC)	ROI (SSC)	eIFA
Bellwether	0.178	38.6	0.108	95.6	0.002
EASC_E	0.141	39.6	0.131	92.5	0.011
EASC_NE	0.251	34.7	**0.151**	**128.9**	**0.000**
SC	0.129	38.2	0.081	75.3	0.007
CLA	0.206	34.0	0.136	122.9	**0.000**
FCM	0.040	31.0	0.085	56.7	0.003
ManualDown	**0.268**	29.0	0.142	125.5	**0.000**
ManualUp	-0.150	**102.8**	-0.282	17.3	0.084
ONE	0.219	33.4	0.145	108.0	**0.000**
	Mean (standard deviation)				
Bellwether	0.175(0.163)	56.3(59.1)	0.106(0.117)	145.3(162.8)	0.020(0.042)
EASC_E	0.144(0.138)	73.8(132.3)	0.122(0.121)	149.4(170.0)	0.023(0.042)
EASC_NE	0.253(0.128)	53.1(53.3)	**0.157(0.116)**	**206.6(215.2)**	0.019(0.042)
SC	0.144(0.151)	74.1(141.0)	0.089(0.119)	140.0(168.8)	0.027(0.048)
CLA	0.217(0.133)	62.2(74.6)	0.137(0.119)	160.4(166.3)	**0.017(0.035)**
FCM	0.037(0.177)	58.6(89.4)	0.074(0.147)	132.8(172.8)	0.028(0.056)
ManualDown	**0.284(0.137)**	47.3(45.6)	0.141(0.116)	202.2(218.7)	0.024(0.048)
ManualUp	-0.167(0.097)	**2268.7(11500.2)**	-0.285(0.124)	52.6(93.5)	0.118(0.115)
ONE	0.240(0.131)	57.0(57.7)	0.150(0.111)	170.0(187.1)	0.018(0.039)

The best performance value of each indicator is marked in **bold** while the worst performance value is marked in underlined.

which is close to EASC_NE (median value of 0.151). The rank comparison for ROI indicates that ONE is better than EASC_NE under SNM but worse under SSC. The median and mean eIFA values of ONE and EASC_NE are similar.

Overall, ONE demonstrates strong prediction performance under both SNM and SSC, considering both context switch effort and code inspection effort, particularly in terms of ROI and eIFA. Among the compared models, EASC_NE and ManualDown show promise as competitors to ONE. However, they have inferior performance in terms of ROI under SNM. Additionally, EASC_NE, being a supervised model, exhibits unstable performances dependent on the training data used, as evidenced by the MCC performance distributions in Fig. 8. This instability makes it unsuitable as a stable global reference point for comparing model performance across studies.

Answer for RQ2: Besides its simplicity and stability, ONE also exhibits a strong defect prediction performance, regardless of whether the SNM or SSC setting is employed. The three key attributes of ONE—simplicity in implementation, strong predictive ability, and stable prediction performance, making it a suitable choice as a common baseline model in MATTER.

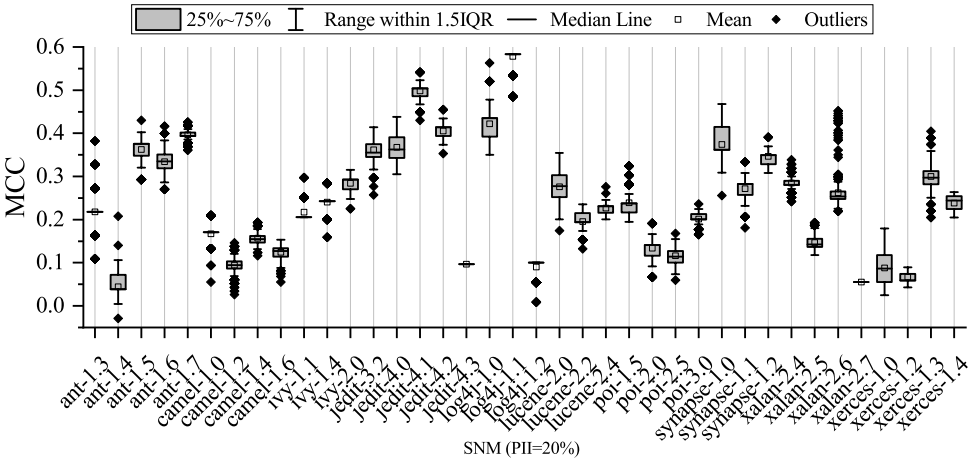


Fig. 8. MCC performance distributions of EASE_NE under PII = 20% on the test set (i.e., the x-axis) when randomly sampling half sets from all available training sets 500 times in the JURECZKO dataset as the training data.

6.3. RQ3: To what extent has progress been made in defect prediction according to MATTER?

In line with Fig. 7, Fig. 9 shows the grouped performance ranking distributions of both state-of-the-art models and ONE. Additionally, Table 9 presents a summary of the median and mean values of three indicators for both SNM (PII = 20%) and SSC (PCI = 20%). The detailed comparison results on individual test datasets can be found in the online Appendix D [18]. It is worth noting that these results further reinforce the conclusion drawn from the analysis of all the datasets, thus providing consistent findings across the evaluation.

Based on Fig. 9 and Table 9, the following observations can be made:

- *CamargoCruz09NB, Amasaki15-NB, and Peters15-NB demonstrate similar performance to ONE in terms of MCC and ROI, but they are inferior in terms of eIFA.* Under SNM, these representative models achieve comparable median MCC values (ranging from 0.209 to 0.222) to ONE's median MCC of 0.219, and they have a higher median ROI (ranging from 37.2 to 40.0 compared to ONE's 33.4). However, as shown in Fig. 9(a), their differences in performance rankings are negligible. Under SSC, these models achieve higher median MCC values (ranging from 0.155 to 0.160) compared to ONE's median MCC of 0.145, and they also exhibit higher median ROI (ranging from 120.5 to 122.5) compared to ONE's 108.0. However, as indicated in Fig. 9(b), the performance ranking differences among these models remain negligible. Notably, ONE significantly outperforms these models in terms of eIFA.
- *KSETE demonstrates similar performance to ONE in terms of ROI under SNM, but it is inferior to ONE in all other cases.* Under SNM, KSETE achieves a better median ROI (39.9) compared to ONE's median ROI of 33.4. However, as shown in Fig. 8(a), the differences in performance rankings between KSETE and ONE are negligible. In terms of MCC, ONE outperforms KSETE significantly, with a statistically significant difference and a non-negligible effect size. Under SSC, ONE outperforms KSETE in both MCC and ROI. Additionally, ONE significantly outperforms KSETE in terms of eIFA.

In our online Appendix B [18], the experimental results reveal that both top-core and MSMDA models also fail to outperform ONE in terms of practical application. Furthermore, we conduct a case study to quantitatively analyze why ONE performs well under effort-aware evaluation in Appendix F [18], and according to this priliminary case study, ONE saves the most effort in finding the first defective module compared to three SOTA models.

Answer for RQ3: According to the findings from MATTER, none of the models investigated outperforms ONE in terms of defect prediction under both SNM and SSC settings, at least for the datasets we utilized. This suggests that the reported progress in defect prediction, as documented in the literature, may not translate into practical effectiveness. It becomes apparent that without employing a consistent evaluation framework like MATTER, it is difficult to assess the true extent of progress achieved in defect prediction.

7. Discussion

In this section, we discuss our findings, including the influence of important factors, implications for defect prediction, and threats to validity.

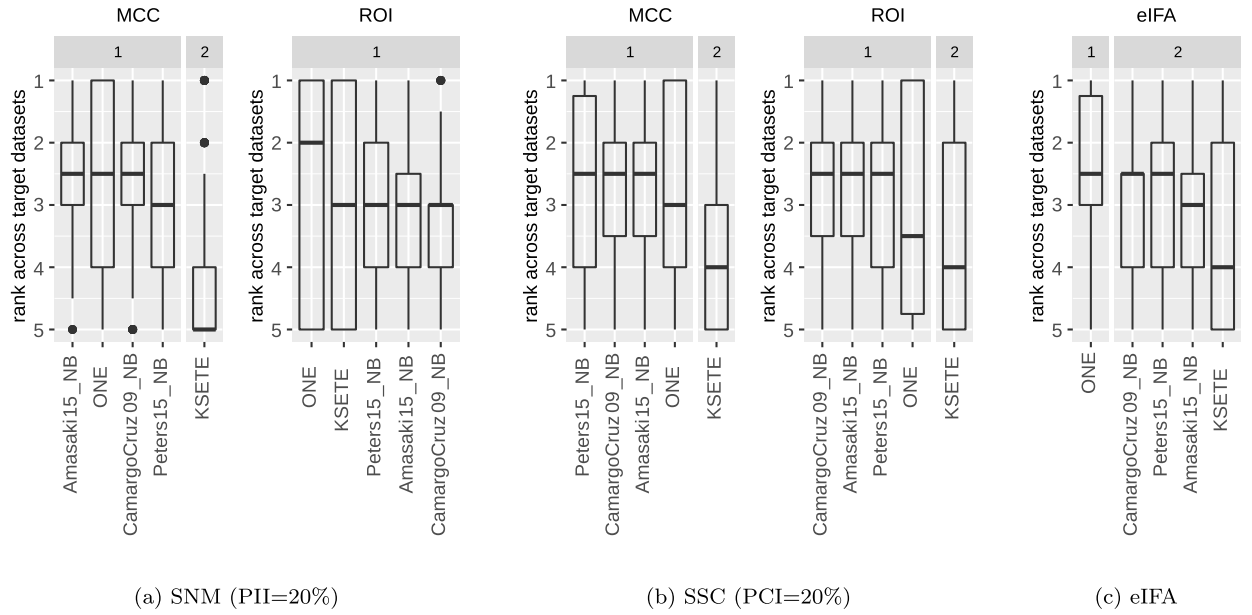


Fig. 9. The performance ranking distributions of different models by the NP SK-ESD test (The smaller the group number on the top of the plot, the better the performance ranking is).

Table 9
The median value and mean (standard deviation) value of representative models and ONE under MATTER.

	Median				
	MCC (SNM)	ROI (SNM)	MCC (SSC)	ROI (SSC)	eIFA
KSETE	0.145	39.9	0.108	97.2	0.013
CamargoCruz09-NB	0.217	38.2	0.157	120.5	0.009
Amasaki15-NB	0.222	37.2	0.155	122.0	0.011
Peters15-NB	0.209	**40.4**	**0.160**	**122.5**	0.008
ONE	0.219	33.4	0.145	108.0	**0.000**
	Mean (standard deviation)				
KSETE	0.146(0.093)	**85.1(185.0)**	0.094(0.079)	160.8(173.3)	0.019(0.028)
CamargoCruz09-NB	0.229(0.125)	57.4(63.1)	**0.155(0.111)**	**201.2(215.5)**	0.021(0.033)
Amasaki15-NB	0.233(0.123)	57.5(62.9)	**0.155(0.104)**	198.6(207.0)	0.022(0.032)
Peters15-NB	0.216(0.146)	59.8(64.4)	0.154(0.136)	190.3(198.8)	0.022(0.041)
ONE	**0.240(0.131)**	57.0(57.7)	0.150(0.111)	170.0(187.1)	**0.018(0.039)**

The best performance value of each indicator is marked in **bold** while the worst performance value is marked in underlined.

7.1. The influence of SQA-effort-aligned threshold

Fig. 10 presents the changes in the median MCC and ROI of SDP models as the code inspection effort/context switch effort increases under SNM/SSC. For comparison, we selected CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB, as these models demonstrated good performance in RQ3.

In Fig. 10(a), we observe that the relative rank of ONE remains stable compared to these models under MATTER when the effort threshold ranges from 0% to 50% PCI for CamargoCruz09-NB and Amasaki15-NB, and up to 70% for Peters15-NB. Similarly, Fig. 10(b) shows consistent comparison results for the three supervised models when the effort threshold varies from 0% to 50% PII. When considering these three representative models, we find that their rankings of the modules in the test projects are not significantly superior to ONE, particularly for the top-ranked modules.

In conclusion, the comparison results remain relatively stable when the available effort is less than 50% PII or 50% PCI. In practical scenarios, it is common to have limited inspection and testing resources for software quality assurance. Therefore, varying the SQA-effort-aligned threshold in MATTER does not have a substantial impact on the overall conclusion.

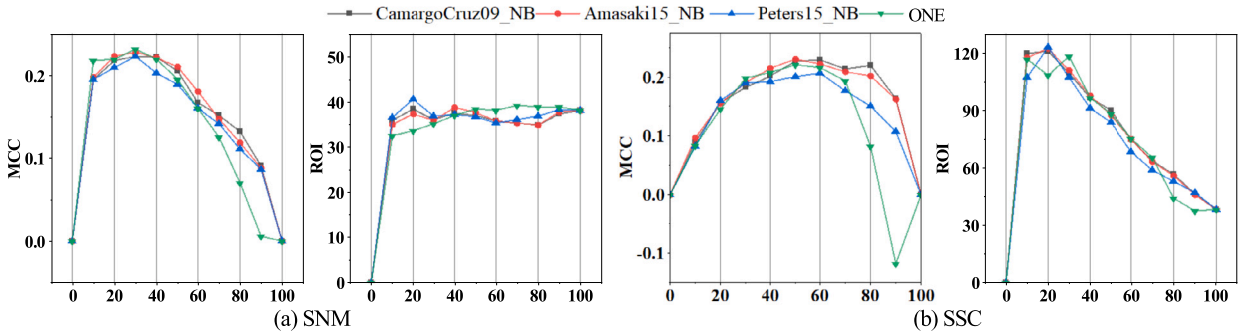


Fig. 10. The median performance trend of evaluated models and ONE varies with the SQA-effort threshold (under SNM and SSC).

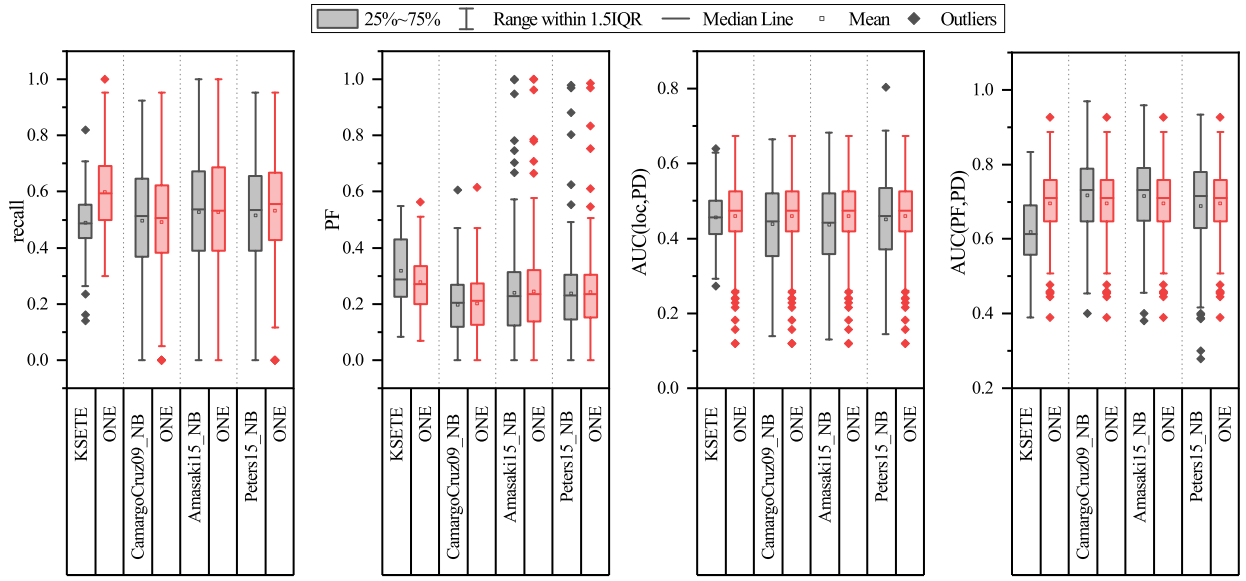


Fig. 11. The performance comparison of ONE with the representative models under the traditional performance evaluation setting.

7.2. How well does ONE perform under the traditional performance evaluation setting?

In RQ3, we find that ONE is a strong baseline under the setting of MATTER. It is natural to ask how well ONE performs under the traditional evaluation settings. To answer this problem, we conduct a comparison of ONE with the representative models under the following setting: (1) use recall, PF, AUC(loc, PD), and AUC(PF, PD); and (2) use the default classification threshold for each representative model. Here, PF denotes the probability of false alarm and PD denotes the recall. AUC(loc, PD) denotes the area under the loc-vs-PD curve (x-axis is the effort measured by the SLOC percentage that the predicted defective modules account for, y-axis is the corresponding PD), while AUC(PF, PD) denotes the area under the PF-vs-PD curve (x-axis is PF, y-axis is the corresponding PD).

Fig. 11 shows the performance distributions of pairs of the evaluated model and ONE aligned by the required PII of the evaluated model in default in terms of recall, PF, AUC(loc, PD), and AUC(PF, PD). Note that we set a classification threshold so that ONE has the same number of predicted defective modules as each compared model. From Fig. 11, we have the following observations. First, ONE has a competitive performance, regardless of which traditional performance indicator is considered. Second, ONE has a median recall around 0.60, a median PF around 0.3, a median AUC(loc, PD) around 0.5, and a median AUC(PF, PD) around 0.7. Note that ONE is an unsupervised model built using only one single metric. In this sense, ONE is a strong baseline, regardless of whether the relative performance or the absolute performance is considered. In summary, ONE is strong enough in defect prediction to provide a "floor performance", helping quickly filter out any model that falls "below the floor".

7.3. Survey study of developers' perspective

In addition to validating the effectiveness of MATTER through quantitative experiments on a publicly available dataset, we also conducted a qualitative analysis of subjective feedback from developers. In this section, we conduct a survey study of practitioners' perspectives on SDP model evaluation.

Table 10
Survey design on developer preferences for defect prediction models.

ID	Question type	Question content
1	Prolific ID	1.Your Prolific ID
2	Attention check	2.This survey will help us to propose useful automatic software bug prediction tools for developers and researchers. Do you commit to providing thoughtful answers?
3	Demographic information	3.What is your current affiliation? (Industry or Academia)
4	Demographic information	4.Are you currently a student?
5	Comprehension check	5.Which of the following programming languages is used primarily for web development?
6	Comprehension check	6.Which data structure follows the First In Last Out (FILO) principle?
7	Demographic information	7.How long have you been coding?
8	Demographic information	8.What is your highest level of education in the field of software engineering?
9	ROI under SNM	
10	ROI under SNM	
11	eIFA and ROI under SSC	[Preference for inspection] If you must inspect every file in the list (you cannot skip a clean file because you do not know if it is clean or buggy until you finish your inspection), which list do you prefer to inspect from left to right? (If you need to find more buggy files as soon as possible)
12	eIFA and ROI under SNM	
13	eIFA	
14	ROI under SSC	
15	eIFA and ROI under SSC	
16	ROI under SNM	
17	Attention check	Same question with 16
18	SNM and SSC vs. non-alignment	[Preference for comparison] In your opinion, which kind of comparison of defect prediction models is fair? (The comparison aims to decide which prediction model can find more buggy files as soon as possible)
19	SSC vs. non-alignment	
20	SSC vs. SNM	

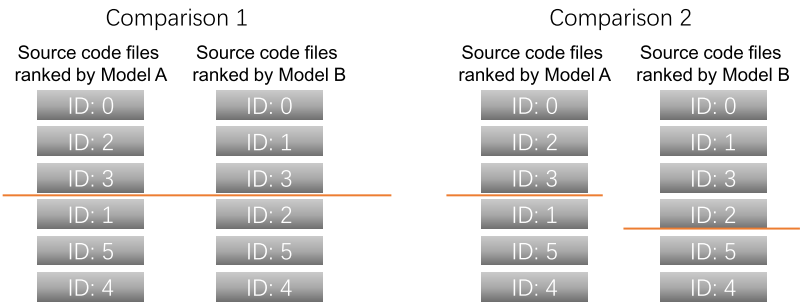


Fig. 12. Options of Question 19 to test participants' preference in SQA-effort threshold alignment.

Survey Objectives. The survey aims to investigate whether models with a better eIFA and ROI performance are preferred by developers, and whether comparisons under aligned SQA effort are preferred by developers. If the answer is yes, it means our MATTER framework is helpful in selecting practical SDP models for developers.

Survey Design. According to this objective, our survey contains four types of questions: demographic information, attention and comprehension check, preference of suggested inspection order of models with different effort-aware performance, and preference of threshold settings in model comparison. As can be seen in Table 10, demographic information (Questions 3, 4, 7, 8) refers to data collected about the characteristics of our respondents. Two attention check questions (Questions 2, 17) are inserted at the beginning and the end of the survey to gauge whether respondents are actively engaged and paying attention. Two comprehension check questions (Questions 5, 6) are put at the beginning of the survey to measure participant's level of understanding of basic software development knowledge to ensure that are our target participants. 11 single-choice questions ask participants about their preferences regarding effort-aware performance indicators (Question 9 to 16) and model comparison threshold settings (Question 18 to 20).

Preferences regarding effort-aware performance indicators. Question 9 gives two options for participants, A.[{LOC:300, actual_label:buggy}] and B.[{LOC:100, actual_label:buggy}]. Option A indicates a smaller ROI under SNM since it contains more LOC. We ask participants which list of files they prefer to inspect, to whether they prefer models with a higher ROI under SNM. If a participant chose Option B, we record one agreement on our indicator ROI under SNM. Questions 10 to 16 adopt a similar design to Question 9.

Preferences regarding model comparison threshold settings. Question 18 shows two comparison settings to compare the two models A and B. As can be seen in Fig. 12, under Comparison 1, Model A and Model B are compared under SSC and SNM, while under Comparison 2, Model A and Model B are compared under unaligned number of modules and LOC. If a participant chose Option A, we record one agreement on our SNM-aligned setting and SSC-aligned setting. Questions 19 and 20 adopt a similar design to Question 18. The complete survey can be found in our online replication kit [18].

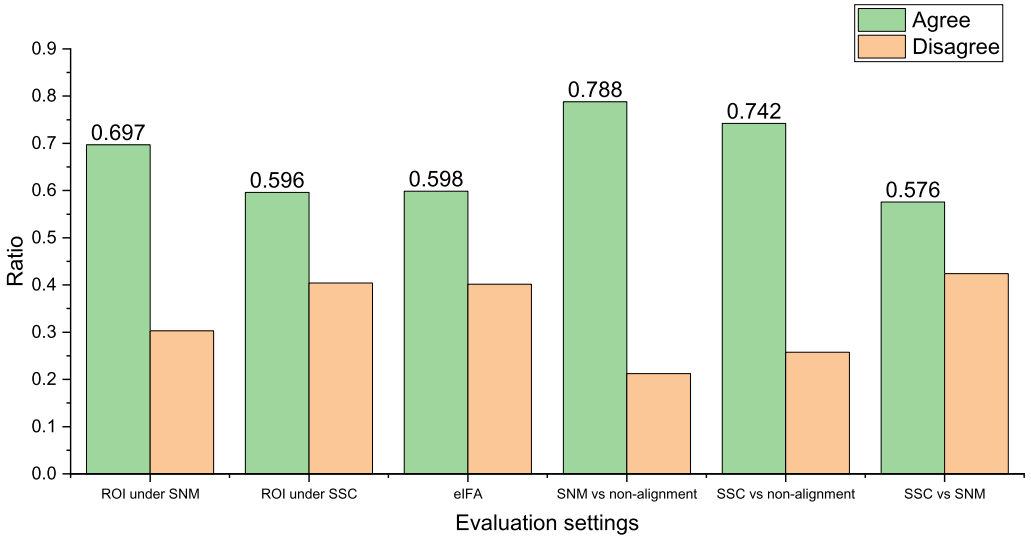


Fig. 13. The survey result of developers' agreement ratio with MATTER's preference on models and model comparisons.

Obtaining Valid Data. The target population of our survey is software developers. To reach the target population, we used a recruiting service provided by Prolific. We rejected 34 participants who failed in attention check or comprehension check and received 33 valid submissions from 15 countries. 19 of them are from the industry and 14 from academia. The average time to finish this survey is 7.5 min. There are 6 participants with less than 1 year of experience, 11 participants with 1-3 years of experience, 9 participants with 3-5 years of experience, and 7 participants with more than five years of experience.

Data analysis. Fig. 13 reports developers' agreement ratio with MATTER's preference on models and model comparisons. When comparing SDP models, 69.7% of answers agree that models with a higher ROI (under SNM) are better, 59.6% of answers agree that models with a higher ROI (under SSC) are better, 59.8% of answers agree that models with a higher eIFA are better. In conclusion, ROI and eIFA are helpful in selecting models that are preferred by software developers. When comparing comparison settings, 78.8% of answers agree that comparing models under SNM is more fair than comparing models without aligning SQA-effort, and 74.2% of answers agree that comparing models under SSC is more fair than comparing models without aligning SQA-effort threshold. In conclusion, comparing SDP models under SSC and SNM is considered fairer than under the unaligned SQA-effort threshold. Moreover, 57.6% of participants believe the SSC threshold is fairer than the SNM threshold while 42.4% hold the opposite view, which indicates both the SNM and SSC settings we advocate are practical in SDP model comparisons. In summary, the result of this survey study supports the effectiveness of MATTER framework from the developers' perspective.

7.4. Implications for defect prediction community

For researchers, our research highlights the need for further advancements in the field of defect prediction, with MATTER serving as a valuable tool for reliable evaluation. Surprisingly, when evaluating representative defect prediction models using MATTER, we discovered that most of them do not outperform the simple baseline model ONE in terms of effort-aware prediction performance. This finding emphasizes the inconsistency in evaluating the performance of defect prediction models within our community. To address this issue, we strongly recommend the continuous use of MATTER with consistent benchmark test datasets for evaluating any newly proposed defect prediction models. This approach will foster consistent performance evaluation and promote the healthy and reliable development of defect prediction techniques. Fig. 14 provides a concrete example of how to use MATTER to investigate the effectiveness of a newly proposed SDP model. Specifically, after proposing a new SDP model, researchers can conduct comparative experiments among SOTA models (if needed) and ONE on certain datasets. If ONE is superior to the new model or the compared SOTA is superior to the new model, then we can say the new model did not improve the prediction performance.

For practitioners, our framework MATTER provides a means to assess the practical value of complex defect prediction models before their application. In recent years, there has been a rise in the use of sophisticated modeling techniques, such as deep learning approaches [69,70], for building defect prediction models. However, these models often require significant computational resources and large amounts of labeled data for training. This presents practical barriers for practitioners, particularly considering the complexity of tuning the numerous parameters involved. Our study highlights the importance of evaluating the practical effectiveness of complex defect prediction models through MATTER. These models should only be employed if they demonstrate significantly better prediction performance compared to the simple model ONE. Otherwise, the anticipated return may not justify the investment. Fig. 15 provides a concrete example of how to use MATTER to decide which SDP model should be incorporated in real-world development. Specifically, if the historical labeled data is not available or there is not enough time to collect training data, then practitioners can directly use ONE as a simple yet strong SDP model. Otherwise, they can collect training data and train some candidate SDP models. After that, MATTER is used to compare the prediction performance of all candidate models and ONE. Finally, the best-performing

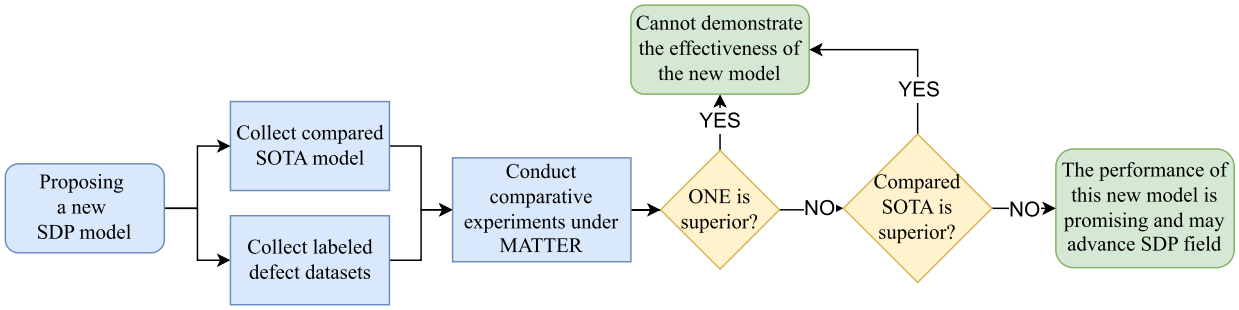


Fig. 14. Actionable example of using MATTER to chose a suitable SDP model for certain software projects by researchers.

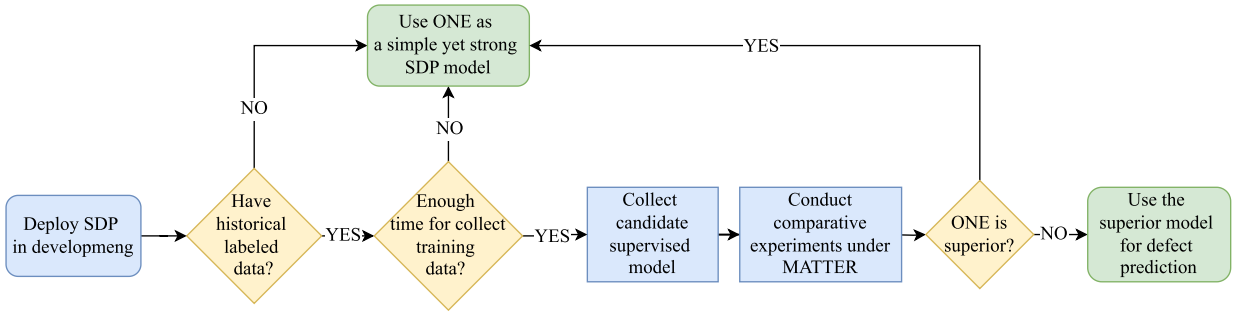


Fig. 15. Actionable example of using MATTER to chose a suitable SDP model for certain software projects by practitioners.

model can be deployed in their development system. Note that practitioners can select and extract different types of metrics such as organizational metrics [23], process metrics [71], and micro interaction metrics [72] to depict the characteristics of their complex projects, rather than only using source code metrics. Such feature selection and extraction process described above is transparent to MATTER. MATTER serves to fairly compare the comparison results of different models on the same batch of test instances, without focusing on the metrics used by compared SDP models. Therefore, MATTER works the same we as we described in complex systems in real-world, because what metrics to choose in complex systems is a matter for the SDP models to consider, not for MATTER to consider.

In summary, our contribution of the MATTER framework promotes consistent performance evaluation in defect prediction. We advocate future research to evaluate any new defect prediction models using MATTER and consistent benchmark test datasets to demonstrate their practical value. This approach will facilitate the development of effective models for predicting defects in software projects, driving advancements in the field.

8. Threats to the validity of this study

Construct validity. The primary threat to our study is the potential mismatch between our measurement of SQA-effort in MATTER and the actual effort required in software quality assurance (SQA). While we adopted the common practice of using PCI and PII to quantify code inspection effort and context switch effort, respectively, the SQA process itself is intricate, and accurately measuring the effort spent on inspected or tested modules is challenging. Nevertheless, our groundbreaking exploration of aligning effort thresholds in comparisons remains significant, and the measurement of code inspection effort and context switch effort in our framework is reasonable, despite its imperfections.

Internal validity. The main threat to our study lies in the possibility that our selection of compared SDP models for RQ1, RQ2, and RQ3 may not be fully representative. However, we have taken steps to minimize this threat by carefully selecting models for each research question. For RQ1 and RQ2, we included both recommended supervised baseline models from the literature and unsupervised models that showed potential as baseline models. This approach ensures a comprehensive representation of different model types. Similarly, for RQ3, we considered not only the best-performing SDP models from a recent comparative study but also state-of-the-art SDP models published in very recent years. By adopting this approach, we have minimized the threat to representativeness as much as possible.

External validity. The main threat to our study is the potential lack of generalizability of our findings to other projects. Our experiments are conducted on 179 releases of five defect datasets, encompassing a diverse range of domains. To mitigate this external threat, we employ the NP SK-ESD test on the ranking distributions of compared models, rather than solely relying on performance indicator values. This approach helps reduce the influence of individual performance values on the comparison results for target releases. While we believe this external threat is within an acceptable range, we acknowledge that our results cannot be universally

applied to all projects. To address this, we have taken steps to enhance the external validity of our study. We have open-sourced our code and data, enabling easy replication of our study in the future [18].

9. Related work

Model building frameworks. Song et al. [73] introduced the GPF (General software defect-proneness Prediction Framework), which focuses on developing an SDP model with high generalization ability by selecting a learning scheme based on historical data. Similarly, Jing et al. [4] proposed the ISDA/SSTCA+ISDA, aiming to address the class-imbalance problem in training data to achieve high prediction performance. However, MATTER serves a different purpose by facilitating an objective assessment of the progress made in defect prediction within our community. The key distinction between MATTER and these frameworks is that GPF and ISDA/SSTCA+ISDA are used for model building, specifically during the model building stage, whereas MATTER is designed for model comparisons during the evaluation stage.

Prediction performance benchmarking. In recent years, there have been significant efforts in benchmarking the performance of various SDP models. Lessmann et al. [74] conducted a benchmarking study comparing the performance of 22 classifiers on the NASA dataset. Ghotra et al. [75] revisited this work using the cleaned NASA dataset and the PROMISE dataset. Herbold et al. [14] conducted a large-scale comparative study benchmarking CPDP models, introducing their benchmark kit called “Crosspare.” Xu et al. [15] conducted a detailed analysis of SDP studies focusing on clustering methods. However, it is important to note that these comparisons were conducted under different threshold settings, which did not align with the SQA effort. Additionally, a “global reference point” for comparison was not established in these studies. Yang et al. [76] compared simple unsupervised models with state-of-the-art supervised ones in effort-aware just-in-time (JIT) defect prediction, revealing superior performance of unsupervised models. The research field of their work is just-in-time defect prediction, which predicts the defectiveness of code commits, allowing developers to proactively identify and address issues during the software development process. However, our work is conducted on identifying defects for modules (e.g., classes, files, methods).

New SDP model designing. A significant portion of defect prediction research aims to propose innovative defect prediction models to enhance predictive performance. For example, Panichella et al. [77] focus on proposing a new search-based SDP model for cost-aware defect prediction; Canfora et al. [78] propose a method named MODEP, which innovatively models the SDP problem as a multi-objective optimization problem and trains it using genetic algorithms; Niederma et al. [79] proposes an inverse defect prediction model, identifying low fault risk methods for deferred testing, improving resource allocation; Bommi et al. [80] conducts comparative experiments to explore classification techniques in the context of SDP, address data imbalance and propose explainable AI integration for model transparency, rather than propose a baseline model for SDP. Our work fundamentally differs from this line of research in two aspects: (1) we propose an evaluation framework for defect prediction models, focusing on how to fairly evaluate the performance of SDP models, while they propose defect prediction models aiming to improve the predictive performance of SDP models, (2) within our framework, we introduce a new baseline model designed to provide a simple, strong and stable baseline, rather than aiming to advance the performance of defect prediction. Moreover, we add extra comparison between ONE vs. [77] and ONE vs. [78] in Appendix G [18], and the comparison results show that ONE performs better than their reported performances, which indicates that ONE is competitive compared to those previous SDP baseline models.

Incorporating defect prediction into industrial frameworks. In previous studies, researchers have investigated integrating SDP technique into the industry from several aspects. For example, Staron et al. [81] provided an automation system to collect software metrics from several large software projects in Ericsson. Their metric measurement framework focuses on metrics definition and automatic collection, while our framework focuses on reliable performance evaluation in defect prediction. Rana et al. [82] proposed a framework to compare SDP models and decide whether an SDP model should be integrated into the industry by a series of checklist based on the Technology Acceptance Model (TAM). Their checklist includes predictive accuracy, running time, cost of license, maintenance cost, etc., while our defect prediction evaluation framework focuses on evaluating SDP models through empirical comparative experiments on labeled datasets. In conclusion, our work mainly conducted comparative experiments on five publicly available datasets to investigate the impact of aligning or SQA-effort for comparing defect prediction models, the effectiveness of our proposed baseline model, and using MATTER to evaluate state-of-the-art SDP models to provide empirical evidence of adopting our research in the industry, which is different with [81] and [82] which mainly take a collaborative approach with developers for qualitative analysis.

10. Conclusion and future work

Defect prediction is a highly active research area, with a multitude of SDP models being developed and proposed. While this contributes to our understanding of leveraging different information for identifying potential defects in software projects, it also presents challenges in evaluating and comparing these models. Currently, the practice of comparing new models against various baselines using diverse indicators lacks a widely accepted global reference point, making it difficult to assess the true progress in defect prediction. Furthermore, many comparisons are conducted without aligning the SQA-effort thresholds, leading to unfair comparisons and potentially misleading conclusions.

To address these issues, our study introduces the MATTER framework, which aims to provide reliable and consistent performance comparisons for SDP models. Using MATTER, we evaluated the effectiveness of representative SDP models and found that none of them significantly outperformed the baseline model ONE when considering SQA effort. This highlights the need for further research and emphasizes the role of MATTER in promoting the healthy and reliable development of defect prediction. For the software

engineering community, MATTER offers a means to evaluate the practical value of complex SDP models before their implementation. The application of MATTER is lightweight, as the ONE baseline model can be easily built using test data independent of the training data. Additionally, the alignment of SQA-effort thresholds and computation of core performance indicators are straightforward. We strongly recommend the adoption of MATTER to evaluate the actual usefulness of SDP models, fostering scientific progress in defect prediction.

In future studies, we plan to expand the investigation of practical effectiveness for additional SDP models using the MATTER framework. This serves the dual purpose of aiding practitioners in identifying truly useful models and assisting researchers in identifying promising modeling techniques for further development and advancement in defect prediction.

CRediT authorship contribution statement

Xutong Liu: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Shiran Liu:** Writing – review & editing, Project administration, Formal analysis, Data curation. **Zhaoqiang Guo:** Writing – review & editing, Project administration, Formal analysis, Data curation. **Peng Zhang:** Writing – review & editing, Project administration, Formal analysis, Data curation. **Yibiao Yang:** Writing – review & editing, Formal analysis. **Huihui Liu:** Writing – review & editing, Formal analysis. **Hongmin Lu:** Writing – review & editing, Formal analysis. **Yanhui Li:** Writing – review & editing, Formal analysis. **Lin Chen:** Writing – review & editing, Formal analysis. **Yuming Zhou:** Writing – review & editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We are very grateful to (1) the authors of Bellwether, SC, CLA, and KSETE for sharing their implementation code; (2) Herbold, Trautsch, and Jens Grabowski for sharing the implementations of CamargoCruz09-NB, Amasaki15-NB, and Peters15-NB; and (3) the authors of FCM and EASC for providing the detail descriptions on their models. In particular, we thank Dr. Ni for illustrating the implementation detail of EASC by communication, ensuring the accurate implementation of EASC in our study. Last but not least, we want to point out that our purpose is not to criticize any existing work but instead to work together as a community to get through the severe and urgent challenges in consistent defect prediction evaluation we face today. This work is partially supported by National Natural Science Foundation of China (62172205, 62172202, 62302222, 62072194).

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.scico.2024.103164.

References

[1] J. Jiarpakdee, C.K. Tantithamthavorn, J. Grundy, Practitioners' perceptions of the goals and visual explanations of defect prediction models, https://doi.org/10.1109/MSR52588.2021.00055, 2021.
[2] Z. Wan, X. Xia, A.E. Hassan, D. Lo, J. Yin, X. Yang, Perceptions, expectations, and challenges in defect prediction, IEEE Trans. Softw. Eng. 46 (11) (2020) 1241–1266, https://doi.org/10.1109/TSE.2018.2877678.
[3] P.R. Bal, S. Kumar, A data transfer and relevant metrics matching based approach for heterogeneous defect prediction, IEEE Trans. Softw. Eng. 49 (3) (2023) 1232–1245, https://doi.org/10.1145/3460319.3464819.
[4] X.-Y. Jing, F. Wu, X. Dong, B. Xu, An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems, IEEE Trans. Softw. Eng. 43 (4) (2017) 321–339, https://doi.org/10.1109/TSE.2016.2597849.
[5] R. Kapur, B. Sodhi, A defect estimator for source code: linking defect reports with programming constructs usage metrics, ACM Trans. Softw. Eng. Methodol. 29 (2) (2020) 1–35, https://doi.org/10.1145/3384517.
[6] Z. Li, X.Y. Jing, X. Zhu, H. Zhang, B. Xu, S. Ying, On the multiple sources and privacy preservation issues for heterogeneous defect prediction, IEEE Trans. Softw. Eng. 45 (4) (2019) 391–411, https://doi.org/10.1109/TSE.2017.2780222.
[7] H. Tong, B. Liu, S. Wang, Kernel spectral embedding transfer ensemble for heterogeneous defect prediction, IEEE Trans. Softw. Eng. 47 (9) (2021) 1886–1906, https://doi.org/10.1109/TSE.2019.2939303.
[8] S. Wang, T. Liu, J. Nam, L. Tan, Deep semantic feature learning for software defect prediction, IEEE Trans. Softw. Eng. 46 (12) (2020) 1267–1293, https://doi.org/10.1109/TSE.2018.2877612.
[9] M. Wen, R. Wu, S.-C. Cheung, How well do change sequences predict defects? Sequence learning from software changes, IEEE Trans. Softw. Eng. 46 (11) (2020) 1155–1175, https://doi.org/10.1109/TSE.2018.2876256.
[10] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (6) (2012) 1276–1304, https://doi.org/10.1109/TSE.2011.103.
[11] Z. Zeng, Y. Zhang, H. Zhang, L. Zhang, Deep just-in-time defect prediction: how far are we?, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2021, Association for Computing Machinery, New York, NY, USA, 2021, pp. 427–438.
[12] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, B. Xu, How far we have progressed in the journey? An examination of cross-project defect prediction, ACM Trans. Softw. Eng. Methodol. 27 (1) (2018) 1, https://doi.org/10.1145/3183339.
[13] R. Krishna, T. Menzies, Bellwethers: a baseline method for transfer learning, IEEE Trans. Softw. Eng. 45 (11) (2019) 1081–1105, https://doi.org/10.1109/TSE.2018.2821670.

[14] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Trans. Softw. Eng. 44 (9) (2018) 811–833, https://doi.org/10.1109/TSE.2017.2724538.
[15] Z. Xu, L. Li, M. Yan, J. Liu, X. Luo, J. Grundy, Y. Zhang, X. Zhang, A comprehensive comparative study of clustering-based unsupervised defect prediction models, J. Syst. Softw. 172 (2021) 110862, https://doi.org/10.1016/j.jss.2020.110862, https://www.sciencedirect.com/science/article/pii/S0164121220302521.
[16] R. Moussa, F. Sarro, On the use of evaluation measures for defect prediction studies, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2022, Association for Computing Machinery, New York, NY, USA, 2022, pp. 101–113.
[17] C. Ni, X. Xia, D. Lo, X. Chen, Q. Gu, Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction, IEEE Trans. Softw. Eng. 48 (3) (2022) 786–802, https://doi.org/10.1109/TSE.2020.3001739.
[18] X. Liu, MATTER, https://github.com/liu906/MATTER, Jul. 2023.
[19] J. Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, John Wiley & Sons, 2005.
[20] R. Moser, W. Pedrycz, G. Succi, A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, in: Proceedings of the 30th International Conference on Software Engineering, ICSE'08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 181–190.
[21] T. Illes-Seifert, B. Paech, Exploring the relationship of a file's history and its fault-proneness: an empirical method and its application to open source programs, Inf. Softw. Technol. 52 (5) (2010) 539–558, https://doi.org/10.1016/j.infsof.2009.11.010, https://www.sciencedirect.com/science/article/pii/S0950584909002109.
[22] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models, Empir. Softw. Eng. 13 (5) (2008) 539–559, https://doi.org/10.1007/s10664-008-9082-8.
[23] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, E. Cialini, Merits of organizational metrics in defect prediction: an industrial replication, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, 2015, pp. 89–98.
[24] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE Trans. Softw. Eng. 31 (4) (2005) 340–355, https://doi.org/10.1109/TSE.2005.49.
[25] Q. Song, M.J. Shepperd, M. Cartwright, C. Mair, Software defect association mining and defect correction effort prediction, IEEE Trans. Softw. Eng. 32 (2) (2006) 69–82, https://doi.org/10.1109/TSE.2006.1599417.
[26] D. Chen, W. Fu, R. Krishna, T. Menzies, Applications of psychological science for actionable analytics, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 456–467.
[27] J. Li, P. He, J. Zhu, M.R. Lyu, Software defect prediction via convolutional neural network, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 318–328.
[28] H. Tu, Z. Yu, T. Menzies, Better data labelling with emblem (and how that impacts defect prediction), IEEE Trans. Softw. Eng. 48 (1) (2020) 278–294.
[29] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, A. Bernstein, The missing links: bugs and bug-fix commits, in: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, pp. 97–106.
[30] S. Liu, Z. Guo, Y. Li, C. Wang, L. Chen, Z. Sun, Y. Zhou, B. Xu, Inconsistent defect labels: essence, causes, and influence, IEEE Trans. Softw. Eng. 49 (2) (2023) 586–610, https://doi.org/10.1109/TSE.2022.3156787.
[31] J. Nam, S. Kim, Clami: defect prediction on unlabeled datasets (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 452–463.
[32] F. Zhang, Q. Zheng, Y. Zou, A.E. Hassan, Cross-project defect prediction using a connectivity-based unsupervised classifier, in: Proceedings of the 38th International Conference on Software Engineering, ICSE'16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 309–320.
[33] S. Hosseini, B. Turhan, D. Gunarathna, A systematic literature review and meta-analysis on cross project defect prediction, IEEE Trans. Softw. Eng. 45 (2) (2019) 111–147, https://doi.org/10.1109/TSE.2017.2770124.
[34] F. Wilcoxon, Individual Comparisons by Ranking Methods, Springer, 1992, pp. 196–202.
[35] R.J. Grissom, J.J. Kim, Effect Sizes for Research: A Broad Practical Approach, Lawrence Erlbaum Associates Publishers, 2005.
[36] Q. Huang, X. Xia, D. Lo, Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction, Empir. Softw. Eng. 24 (5) (2019) 2823–2862, https://doi.org/10.1007/s10664-018-9661-2.
[37] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Inf. Softw. Technol. 59 (2015) 170–190, https://doi.org/10.1016/j.infsof.2014.11.006, https://www.sciencedirect.com/science/article/pii/S0950584914002523.
[38] Y. Qu, Q. Zheng, J. Chi, Y. Jin, A. He, D. Cui, H. Zhang, T. Liu, Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance, IEEE Trans. Softw. Eng. 47 (2) (2021) 348–366, https://doi.org/10.1109/TSE.2019.2892959.
[39] S. Majumder, T. Xia, R. Krishna, T. Menzies, Methods for stabilizing models across large samples of projects (with case studies on predicting defect and project health), in: Proceedings of the 19th International Conference on Mining Software Repositories, MSR'22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 566–578.
[40] R. Moussa, G. Guizzo, F. Sarro, Meg: multi-objective ensemble generation for software defect prediction, in: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM'22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 159–170.
[41] T. Zhang, Y. Yu, X. Mao, Y. Lu, Z. Li, H. Wang, Fense: a feature-based ensemble modeling approach to cross-project just-in-time defect prediction, Empir. Softw. Eng. 27 (7) (2022) 162.
[42] Z. Li, H. Zhang, X.-Y. Jing, J. Xie, M. Guo, J. Ren, Dssdpp: data selection and sampling based domain programming predictor for cross-project defect prediction, IEEE Trans. Softw. Eng. 49 (4) (2023) 1941–1963, https://doi.org/10.1109/TSE.2022.3204589.
[43] H. Song, G. Wu, L. Ma, Y. Pan, Q. Huang, S. Jiang, Adversarial domain adaptation for cross-project defect prediction, Empir. Softw. Eng. 28 (5) (2023) 127.
[44] A.E.C. Cruz, K. Ochimizu, Towards logistic regression models for predicting fault-prone code across software projects, in: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 460–463.
[45] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, D. Cok, Local vs. global models for effort estimation and defect prediction, in: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011, pp. 343–351.
[46] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empir. Softw. Eng. 14 (5) (2009) 540–578, https://doi.org/10.1007/s10664-008-9103-7.
[47] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a fault prediction model to allow inter languagereuse, in: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE'08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 19–24.
[48] P.A. Whigham, C.A. Owen, S.G. Macdonell, A baseline model for software effort estimation, ACM Trans. Softw. Eng. Methodol. 24 (3) (2015) 1–11, https://doi.org/10.1145/2738037.
[49] N. Li, M.J. Shepperd, Y. Guo, A systematic review of unsupervised learning techniques for software defect prediction, Inf. Softw. Technol. 122 (2020) 106287, https://doi.org/10.1016/j.infsof.2020.106287, https://www.sciencedirect.com/science/article/pii/S0950584920300379.
[50] J. Çarka, M. Esposito, D. Falessi, On effort-aware metrics for defect prediction, Empir. Softw. Eng. 27 (6) (2022) 152, https://doi.org/10.1007/s10664-022-10186-7.
[51] J. Yao, M. Shepperd, Assessing software defection prediction performance: why using the matthews correlation coefficient matters, in: Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering, EASE'20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 120–129.

[52] S. Feng, J. Keung, X. Yu, Y. Xiao, K.E. Bennin, M.A. Kabir, M. Zhang, Coste: complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction, Inf. Softw. Technol. 129 (2021) 106432, https://doi.org/10.1016/j.infsof.2020.106432, https://www.sciencedirect.com/science/article/pii/S0950584920301889.
[53] Q. Huang, X. Xia, D. Lo, Supervised vs unsupervised models: a holistic look at effort-aware just-in-time defect prediction, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 159–170.
[54] L. Lavazza, S. Morasca, Comparing phi and the f-measure as performance metrics for software-related classifications, Empir. Softw. Eng. 27 (7) (2022) 185, https://doi.org/10.1007/s10664-022-10199-2.
[55] S. Morasca, L. Lavazza, On the assessment of software defect prediction models via roc curves, Empir. Softw. Eng. 25 (5) (2020) 3977–4019, https://doi.org/10.1007/s10664-020-09861-4.
[56] D. Chicco, V. Starovoitov, G. Jurman, The benefits of the matthews correlation coefficient (mcc) over the diagnostic odds ratio (dor) in binary classification assessment, IEEE Access 9 (2021) 47112–47124, https://doi.org/10.1109/ACCESS.2021.3068614.
[57] D. Chicco, G. Jurman, The matthews correlation coefficient (mcc) should replace the roc auc as the standard metric for assessing binary classification, BioData Min. 16 (1) (2023) 4, https://doi.org/10.1186/s13040-023-00322-4.
[58] D. Chicco, G. Jurman, The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation, BMC Genomics 21 (1) (2020) 1–13.
[59] S. Herbold, A. Trautsch, J. Grabowski, Correction of “a comparative study to benchmark cross-project defect prediction approaches”, IEEE Trans. Softw. Eng. 45 (6) (2019) 632–636, https://doi.org/10.1109/TSE.2018.2790413.
[60] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: some comments on the nasa software defect datasets, IEEE Trans. Softw. Eng. 39 (9) (2013) 1208–1215.
[61] M. D’Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches, in: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010, pp. 31–41.
[62] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE’10, Association for Computing Machinery, New York, NY, USA, 2010.
[63] R. Wu, H. Zhang, S. Kim, S.-C. Cheung, Relink: recovering links between bugs and changes, in: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE’11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 15–25.
[64] S. Herbold, A. Trautsch, F. Trautsch, B. Ledel, Problems with szz and features: an empirical study of the state of practice of defect prediction data collection, Empir. Softw. Eng. 27 (2) (2022) 1–49, https://doi.org/10.1007/s10664-021-10092-4.
[65] C. Tantithamthavorn, Scottknottesd: The Scott-Knott Effect Size Difference (esd) Test, R package version 2, 2017.
[66] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, The impact of automated parameter optimization on defect prediction models, IEEE Trans. Softw. Eng. 45 (7) (2018) 683–711, https://doi.org/10.1109/TSE.2018.2794977.
[67] N. Cliff, Ordinal Methods for Behavioral Data Analysis, Psychology Press, 2014.
[68] J. Romano, J.D. Kromrey, J. Coraggio, J. Skowronek, Appropriate statistics for ordinal level data: should we really be using t-test and cohen’sd for evaluating group differences on the nsse and other surveys, in: Annual Meeting of the Florida Association of Institutional Research, vol. 177, 2006, p. 34.
[69] C. Manjula, L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics, Clust. Comput. 22 (4) (2019) 9847–9863, https://doi.org/10.1007/s10586-018-1696-z.
[70] F. Dong, J. Wang, Q. Li, G. Xu, S. Zhang, Defect prediction in Android binary executables using deep neural network, Wirel. Pers. Commun. 102 (3) (2018) 2261–2285, https://doi.org/10.1007/s11277-017-5069-3.
[71] F. Rahman, P. Devanbu, How and why, process metrics are better, in: 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 432–441.
[72] T. Lee, J. Nam, D. Han, S. Kim, H. Peter In, Developer micro interaction metrics for software defect prediction, IEEE Trans. Softw. Eng. 42 (11) (2016) 1015–1035, https://doi.org/10.1109/TSE.2016.2550458.
[73] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Trans. Softw. Eng. 37 (3) (2010) 356–370, https://doi.org/10.1109/TSE.2010.90.
[74] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496, https://doi.org/10.1109/TSE.2008.35.
[75] B. Ghotra, S. McIntosh, A.E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, 2015, pp. 789–800.
[76] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung, Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Association for Computing Machinery, New York, NY, USA, 2016, pp. 157–168, https://dl.acm.org/doi/10.1145/2950290.2950353.
[77] A. Panichella, C.V. Alexandru, S. Panichella, A. Bacchelli, H.C. Gall, A search-based training algorithm for cost-aware defect prediction, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO’16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 1077–1084.
[78] G. Canfora, A.D. Lucia, M.D. Penta, R. Oliveto, A. Panichella, S. Panichella, Defect prediction as a multiobjective optimization problem, Softw. Test. Verif. Reliab. 25 (4) (2015) 426–459, https://doi.org/10.1002/stvr.1570, https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1570, https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1570.
[79] R. Niedermayr, T. Röhm, S. Wagner, Too trivial to test? An inverse view on defect prediction to identify methods with low fault risk, in: M. Felderer, W. Hasselbring, R. Rabiser, R. Jung (Eds.), Software Engineering 2020, Fachtagung des GI-Fachbereichs Softwaretechnik, 24.-28. Februar 2020, Innsbruck, Austria, in: LNI, vol. P-300, Gesellschaft für Informatik e.V., 2020, pp. 137–138.
[80] N.S. Bommi, A. Negi, A standard baseline for software defect prediction: using machine learning and explainable ai, in: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), 2023, pp. 1798–1803.
[81] M. Staron, W. Meding, C. Nilsson, A framework for developing measurement systems and its industrial evaluation, Inf. Softw. Technol. 51 (4) (2009) 721–737, https://doi.org/10.1016/j.infsof.2008.10.001, https://www.sciencedirect.com/science/article/pii/S0950584908001419.
[82] R. Rana, M. Staron, J. Hansson, M. Nilsson, W. Meding, A framework for adoption of machine learning in industry for software defect prediction, in: A. Holzinger, T. Libourel, L.A. Maciaszek, S.J. Mellor (Eds.), ICSOFT-EA 2014 - Proceedings of the 9th International Conference on Software Engineering and Applications, Vienna, Austria, 29-31 August, 2014, SciTePress, 2014, pp. 383–392.