Expert Systems With Applications 245 (2024) 123041

Contents lists available at ScienceDirect

Expert Systems With Applications

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

On the relative value of clustering techniques for Unsupervised Effort-Aware Defect Prediction

Peixin Yang [a,b], Lin Zhu [c], Yanjiao Zhang [c], Chuanxiang Ma [d,e], Liming Liu [f], Xiao Yu [a,g,*], Wenhua Hu [a]

[a] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China
[b] Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China
[c] School of Computer, Wuhan Qingchuan University, Wuhan, China
[d] School of Computer Science and Information Engineering, Hubei University, Wuhan, China
[e] Hubei Province Project of Key Research Institute of Humanities and Social Sciences at Universities (Research Center of Information Management for Performance Evaluation), Wuhan, China
[f] School of Cyber Science and Engineering, Wuhan University, Wuhan, China
[g] Wuhan University of Technology Chongqing Research Institute, Chongqing, China

ARTICLE INFO

Keywords:
Software defect prediction
Effort-aware
Clustering technique
Unsupervised learning

ABSTRACT

Unsupervised Effort-Aware Defect Prediction (EADP) uses unlabeled data to construct a model and ranks software modules according to the software feature values. Xu et al. (JSS 2021) conducted an exploration of clustering techniques for unsupervised defect prediction and found that several clustering methods exhibit better performance on the F1@20% effort-aware metric. However, their conclusion may not be convincing, as they did not take into account the impact of the Initial False Alarms (IFA) metric on unsupervised EADP. Furthermore, their study did not compare with the state-of-the-art supervised EADP models. To further investigate clustering techniques for unsupervised EADP more comprehensively, we explore the performance of 22 clustering techniques for unsupervised EADP using three classification metrics and six effort-aware metrics. The experimental results demonstrate that (1) the best clustering technique for unsupervised EADP, K-medoids, can significantly reduce the IFA of the ManualUp method to an acceptable range. In contrast, the clustering techniques recommended by Xu et al. exhibit a high IFA value that cannot be deemed acceptable by testing teams; (2) K-medoids performs better than some supervised EADP methods, especially on metrics such as IFA and PMI@20% (Proportion of Modules Inspected when inspecting the top 20% lines of code); (3) better classification performance of clustering techniques could lead to better effort-aware performance. In summary, we recommend using the K-medoids clustering technique for unsupervised EADP and suggest that future research devote more effort to exploring better-unsupervised clustering techniques. In support of reproducibility and future research, we provide the source code used in our study (https://github.com/Andre-Yang816/Clustering4UEADP).

1. Introduction

Software defects are a common occurrence in software development and can lead to impaired functionality, errors, and even system crashes, causing significant inconvenience and losses to users (Feng et al., 2024; Gong, Jiang, & Jiang, 2019; Gong et al., 2022; Jin, 2021; Liang et al., 2021). Therefore, it is crucial for development teams to identify and promptly address potential defect issues. Software defect prediction has become an important research direction in the field of software engineering (Feng et al., 2021; Gong, Jiang, Wang, & Jiang, 2019; Li et al., 2021; Majd et al., 2020; Pandey et al., 2020; Shao et al., 2018; Yu et al., 2022, 2018; Zain et al., 2023). By analyzing and mining data during the software development process, EADP can detect potential defects in the early stages of software development and provide corresponding improvement suggestions for development teams, thus improving software quality and development efficiency (Gong et al., 2021; Xiang et al., 2018). Classification-Based Defect Prediction (CBDP) is a software defect prediction approach that uses machine learning classification algorithms to predict whether a software module is likely

* Corresponding author at: School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China.
E-mail addresses: peixinyang@whut.edu.cn (P. Yang), linzhu_cs@126.com (L. Zhu), yanjiaozhang_cs@163.com (Y. Zhang), mcx838@hubu.edu.cn (C. Ma), liming.liu@whu.edu.cn (L. Liu), xiaoyu@whut.edu.cn (X. Yu), whu10@whut.edu.cn (W. Hu).

https://doi.org/10.1016/j.eswa.2023.123041
Received 24 May 2023; Received in revised form 7 December 2023; Accepted 24 December 2023
Available online 28 December 2023

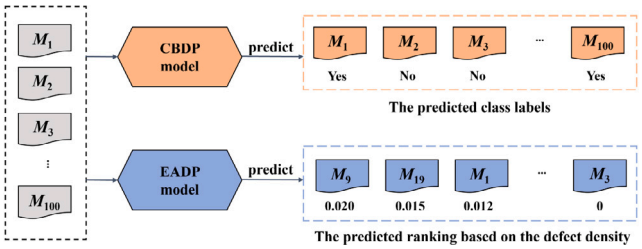

Fig. 1. The differences between CBDP and EADP.

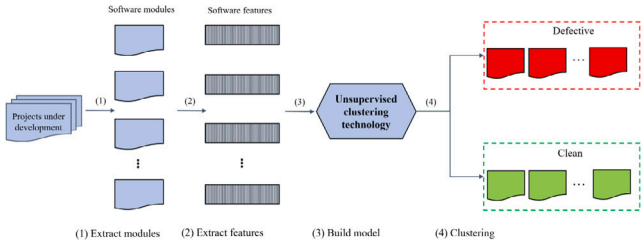

Fig. 2. The process of unsupervised clustering defect prediction.

to contain defects or not (Shao et al., 2018). However, the conventional CBDP research has not taken into account issues related to resource allocation and the cost of defects fix, which may be unable to guide development teams to make optimal decisions.

To address these shortcomings, Mende and Koschke (2010) incorporated effort into the EADP and proposed the Effort-Aware Defect Prediction (EADP), which sorted software modules based on defect density and prioritizes inspection of modules with higher defect densities. The EADP model can more accurately predict the likelihood of defects and provide development teams with more practical recommendations and decision-making support. Software development teams can find more defects when checking a certain number of Lines Of Code (LOC) (Li, Yang, et al., 2023; Yu et al., 2023).

Fig. 1 illustrates the differences between CBDP and EADP. Suppose software testers evaluate a newly developed software system comprising 100 software modules (i.e., M_1, M_2, M_3, ..., M_{100}). The LOC in this system amount to 10,000. However, owing to restricted testing resources, the testers can only focus on a part of the code (e.g., 20% LOC of the entire system). Therefore, they have the option to build either a CBDP model or an EADP model based on historical software data. Suppose the CBDP model predicts that 30 modules with 2,500 LOC are defective. They need to determine which of these modules to inspect first. However, they can test the first several modules (i.e., M_9, M_{19}, M_1, ..., and so on) in descending order of the density provided by the EADP model, until 2,000 LOC are inspected. Therefore, EADP can assist in allocating the limited testing resources more efficiently.

1.1. Motivation

However, EADP typically requires a large amount of labeled data from the current project or external projects to train supervised learning models, which may not always be available (Li, Lu, et al., 2023). Furthermore, the cost of acquiring and annotating defect data is prohibitively high. Unsupervised defect prediction methods do not require manually labeled defect data, but instead employ the data in the development process to discover potential defects (Li et al., 2020). Fig. 2 illustrates the process of unsupervised clustering defect prediction. In detail, the process involves initially extracting modules from the projects under development. Following the extraction, features are extracted from these modules and used to construct the unsupervised clustering model. This model partitions the modules into two clusters: defective and clean modules.

Menzies et al. (2010) proposed an unsupervised model named ManualUp, which arranged modules in descending order based on the inverse of their respective feature values. ManualUp outperforms some supervised models on several effort-aware metrics. However, according to Yang et al. (2016), ManualUp can result in a very high value of the Initial False Alarms (IFA), which can frustrate the software testing team if the IFA value exceeds 10 (Kochhar et al., 2016). Therefore, in subsequent studies on EADP, few scholars have proposed the utilization of unsupervised learning approaches.

However, there have been a lot of studies utilizing unsupervised clustering techniques for CBDP. For example, Zhang et al. (2016) proposed a connectivity-based clustering algorithm, which divided software modules into two clusters using spectral clustering, then labeled each cluster as a defective or clean cluster according to the average row sum of all modules within each cluster. Recently, Xu, Li, et al. (2021) have explored the performance of some clustering techniques for unsupervised EADP using both classification and effort-aware metrics, which employs clustering techniques to divide software modules into two different groups and predict defects based on specific rules according to the characteristics of each group. Their study revealed several clustering techniques that perform well in terms of the F1@20% effort-aware metric, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points To Identify Clustering Structure), AP (Affinity Propagation), and ROCK (RObust Clustering using linKs). However, these clustering techniques all have high IFA values in our research. Therefore, their conclusions may lack persuasiveness because these clustering techniques only hold practical significance when the IFA is less than 10. Furthermore, their study did not compare with the state-of-the-art supervised EADP model, so it remains uncertain about the extent to which unsupervised clustering techniques can match the supervised models.

1.2. Our work and contributions

Inspired by these limitations, we conduct a study and comparison of 22 clustering techniques for unsupervised EADP. Similar to the study by Xu, Li, et al. (2021), we initially use these clustering techniques to divide software modules into defective and clean groups, and subsequently rank the modules based on each feature value, following the approach of ManualUp (Menzies et al., 2010). We compare these unsupervised clustering techniques with the unsupervised method ManualUp and the state-of-the-art supervised EADP methods (including EALR (Kamei et al., 2012), EATT (Li et al., 2020), and CBS+ (Huang et al., 2019)) on the PROMISE dataset. PROMISE consists of 41 releases from 11 open-source software projects. We utilize PofB@20% (Proportion of the found Bugs when inspecting the top 20%LOC) and Recall@20% to evaluate the impact of these 26 methods. Additionally, we employ Precision@20% and IFA to assess the false positive rate, and PMI@20% (Proportion of Modules Inspected when inspecting the top 20% LOC) to quantify the number of software modules that need an inspection. We also employ the Scott-Knott Effect Size Difference (ESD) test and the Wilcoxon signed-rank test to evaluate these methods.

Our research process and experimental results can be condensed into the following four Research Questions (RQs):

RQ1: Which unsupervised clustering technique exhibits the most exemplary performance?

In order to investigate which unsupervised clustering technique exhibits the most exemplary effort-aware performance on the PROMISE dataset, we compare the 22 unsupervised clustering techniques. In detail, we first apply the 22 clustering techniques to cluster the testing dataset and sort the clustering results in ascending order for each feature dimension. Then, we evaluate the modules in the top 20% LOC, utilizing the Scott-Knott ESD test and presenting the plots to demonstrate the statistically distinct groups with significant differences.

The results show that K-medoids, Kmeans++, and CURE (Clustering Using REpresentative) exhibit the most comprehensive effort-aware performance, and it is advisable to avoid utilizing DBSCAN, OPTICS, AP, and BANG for unsupervised EADP.

RQ2: Could unsupervised clustering techniques enhance the performance of the unsupervised EADP model ManualUp?

Previous research (Yu, Bennin, et al., 2019) has shown that ManualUp performs poorly in terms of the IFA metric, with IFA exceeding 10 on multiple PROMISE datasets, which is considered unacceptable (Kochhar et al., 2016). Therefore, we conduct a comprehensive comparison between the top-performing unsupervised clustering techniques (i.e., K-medoids, K-means++, and CURE) and ManualUp to explore whether unsupervised clustering techniques can improve the performance of ManualUp. We utilize the Wilcoxon signed-rank test to examine the significant differences in terms of the metrics.

Our experiment results show that K-medoids, K-means++, and CURE can significantly decrease the IFA and PMI@20% metrics of ManualUp, while only marginally compromising the performance of other effort-aware metrics.

RQ3: How does the performance of unsupervised clustering techniques compare to that of the state-of-the-art supervised EADP model?

Since (Xu, Li, et al., 2021) has not explored the extent to which unsupervised clustering techniques can match the supervised models, we compare K-medoids, K-means++, and CURE with the state-of-the-art supervised EADP methods, including EALR (Effort-Aware Linear Regression (Kamei et al., 2012)), EATT (Effort-Aware Tri-Training (Li et al., 2020)), CBS+ (Classify Before Sorting (Huang et al., 2019)), CBS+(RF) (CBS+ with Random Forests (Balaram & Vasundra, 2022)), CBS+(GB) (CBS+ with Gradient Boosting (Sandhu & Batth, 2021)), and SERS (Shivkumar S., E James W., Ram A. and Sunghun K. Shivaji et al. (2012)).

Our experiment results indicate that unsupervised clustering techniques perform better than some supervised EADP methods, especially on metrics such as IFA and PMI@20%. Although there is still a gap compared to the best supervised EADP method on some metrics, unsupervised clustering techniques do not always perform worse.

RQ4: Is the effort-aware performance of unsupervised clustering techniques related to the classification performance?

We initially employ K-medoids to classify the modules into defective and clean, and subsequently rank the software modules based on each feature. This prompts us to wonder whether the superior classification performance of these clustering methods can lead to better effort-aware performance. We apply the Kendell correlation coefficient to evaluate the correlation among the evaluation metrics and use a heat map to display the results.

The experimental findings reveal a pronounced correlation between classification metrics and effort-aware metrics. This signifies that better classification performance could potentially lead to better effort-aware performance.

RQ5: Can the best-performing unsupervised clustering method be generalized to other datasets?

To explore the findings on the PROMISE dataset, specifically evaluating whether the identified optimal unsupervised clustering method, K-medoids, is equally applicable to other software defect datasets, we conduct comparative experiments on the NASA and SOFTLAB datasets.

The experimental results indicate that the best-performing unsupervised clustering method still exhibits the best performance among these 16 software projects from the NASA and SOFTLAB datasets, compared to other unsupervised clustering methods.

Our contributions are succinctly summarized as follows:

- Integration of Effort-Aware Metrics: Our research emphasizes the consideration of the Initial False Alarms (IFA) metric, which was not taken into account in Xu, Li, et al. (2021) exploring clustering techniques for unsupervised EADP. By evaluating clustering methods using three classification metrics and six effort-aware metrics, including IFA, we provide a more comprehensive analysis of their performance.
- Comparison with Supervised EADP Models: Unlike (Xu, Li, et al., 2021), we conduct a thorough comparison of the clustering techniques for unsupervised EADP with state-of-the-art supervised EADP models. The comparative analysis results indicate that the best-performing unsupervised clustering method, K-medoids, outperforms some supervised methods on IFA and PMI@20%.
- Impact of Classification Performance: We highlight the correlation between the classification performance of clustering techniques and their effort-aware performance. This linkage emphasizes the importance of accurate classification in achieving better effort-aware metrics, showcasing the significance of the selected clustering technique.
- Recommendations for Future Research: Based on our findings, we recommend the use of the K-medoids clustering technique for unsupervised EADP. Additionally, we suggest that future research should focus on exploring improved unsupervised clustering techniques, thereby providing a direction for further investigation and advancement in the field.

1.3. Organization

Section 2 introduces the related work on EADP and clustering techniques for unsupervised EADP. Section 3 provides a brief description of the unsupervised clustering techniques and several EADP methods. Section 4 outlines the setup of our experiment. Section 5 analyzes the experimental results in detail. Sections 6 and 7 respectively present the threats to validity and the implications of our work. Section 8 summarizes our work and draws a conclusion.

2. Related work

2.1. Effort-aware defect prediction

EADP methods prioritize the inspection of modules with higher defect density, thereby improving the efficiency of software resting. The notion of "Effort-Aware" was first introduced by Mende and Koschke (2010) and Kamei et al. (2010) in the field of EADP, where they proposed the EADP technique with the aim of detecting more defects within the same LOC. Several EADP techniques have been proposed in the previous literature, including supervised and unsupervised methods. The EALR model proposed by Kamei et al. (2012), employed linear regression to construct an EADP model, which assisted developers in efficiently reviewing defects with a fixed inspection budget. Yang et al. (2016) proposed an unsupervised model called ManualUp for Just-In-Time (JIT) EADP. The results showed that ManualUp was more effective than some supervised methods on Recall@20%. Bennin, Toda, et al. (2016) and Yu, Bennin, et al. (2019) explored the optimal EADP algorithms, while (Bennin, Keung, et al., 2016) also assessed the influence of data re-sampling techniques on EADP. Fu and Menzies (2017) refuted the conclusions of Yang et al. (2016) and proposed a supervised effort-aware JIT defect prediction model called OneWay based on the ManualUp (Yang et al., 2016). OneWay initially evaluated the unsupervised models from ManualUp using labeled training data and selected the one with the best Recall@20% for prioritizing changes in testing data. The experiments demonstrated that OneWay outperformed most unsupervised models in effort-aware metrics, and a combination of unsupervised learners may achieve comparable performance to supervised learners on a project-by-project basis. Yan et al. (2017) shared similar conclusions with Fu and Menzies (2017) and

pointed out that unsupervised models did not perform significantly better than state-of-the-art supervised models using within-project setup, while unsupervised models can outperform state-of-the-art supervised models significantly using cross-project setup. In addition, the number of files to be inspected should be considered rather than only considering LOC, when evaluating effort-aware file-level defect prediction models. Chen et al. (2018) proposed an EADP method named MULTI, which aimed to maximize recall value and minimize inspection effort, to guide the selection of modules for inspection. It employed a multi-objective optimization algorithm to identify the Pareto-optimal set of solutions, and used a decision-making process to select the most appropriate solution based on the trade-off between two objectives. Huang et al. (2018, 2019) reviewed the works of the supervised EADP model (i.e., EALR Kamei et al., 2012) and unsupervised EADP model (i.e., ManualUp Yang et al., 2016) and pointed out that software testers would encounter lots of initial false alarms (i.e., the high IFA value) and must inspect many software modules (i.e., the high PMI@20% value) according to the rankings of ManualUp. Therefore, they proposed the CBS+ method, including classification and ranking strategy for modules. The experiment results showed that CBS+ could find 15%–26% more faulty modules than EALR and reduce the PMI@20% and IFA values compared with ManualUp. Ni, Xia, Lo, Chen, and Gu (2022), Ni, Xia, Lo, Yang, and Hassan (2022) indicated the superiority of CBS+ for cross-project EADP and JIT EADP on JavaScript projects, respectively. Qu et al. (2021, 2019) have introduced a technique that enhanced the effectiveness of EADP by utilizing k-core decomposition applied to software class dependency networks and developer information. Additionally, Yang et al. (2021) proposed a differential evolution algorithm-based EADP method, and Çarka et al. (2022) proposed to evaluate the EADP performance using the normalized proportion of the found defects, which ranked software modules based on predicted defect densities. In recent decades, the traditional approach to defect prediction at the module level (such as file or class level) has been dominated by supervised models, while applying supervised models in practice can be expensive for practitioners, as collecting defect data is often time-consuming and costly. Furthermore, the supervised EADP models require labeled modules of historical data from either the current project or external projects, which may not always be available. In contrast, unsupervised methods have a low modeling cost and a wide range of applications, as they do not require defect data to build prediction models. Menzies et al. (2010) proposed the unsupervised model ManualUp, which demonstrated superior performance in some effort-aware metrics compared to some supervised models. Nevertheless, if the value of the IFA exceeds 10, it can be frustrating for the software testing team, and ManualUp has the potential to yield such high IFA values, as noted by Kochhar et al. (2016). Therefore, in subsequent studies on unsupervised EADP, few scholars have explored the use of unsupervised learning approaches.

2.2. Clustering techniques for unsupervised classification-based defect prediction

The use of unsupervised methods for CBDP has become a research hotspot in recent years, and numerous studies have been conducted on utilizing unsupervised clustering techniques for CBDP. Unsupervised clustering methods for CBDP generally classify the modules into two clusters, where the modules in one cluster are predicted as defective, while those in the other cluster are predicted as clean. According to Bishnu and Bhattacherjee (2011), a quad tree-based K-means algorithm was utilized to predict defects in program modules. Quad trees were utilized to determine the initial cluster centers to be input to the A'-Means algorithm and it was applied for predicting defects in program modules. Park and Hong (2014) constructed unsupervised EADP models using clustering algorithms (i.e., EM and X-means), to automatically determine the number of clusters. Öztürk et al. (2015) presented a new defect clustering technique using K-means++ for web page source codes and pointed out that linear discriminant analysis performs better than the other three classifiers in general after clustering. Zhang et al. (2016) proposed a connectivity-based classifier, spectral clustering, for cross-project EADP, which ranked as one of the top classifiers, along with five widely-used supervised classifiers (random forest, naive Bayes, logistic regression, decision tree, and logistic model tree) and four other unsupervised classifiers (K-means, partition around medoids, fuzzy c-means and neural-gas) in their experiments. Local models were introduced by Menzies et al. (2011) for EADP. With local models, the available data is first clustered into homogeneous regions, and afterward, separate classifiers are trained for each homogeneous region. Herbold et al. (2017) and Menzies et al. (2012) compared the effects of global models and local models on cross-project EADP. In the local models, clustering techniques such as EM and WHERE were used to cluster the training project set. The experimental results showed that the clustered dataset had better performance. Ha et al. (2019) replicated the experimental results of Zhang et al. (2016) and attempted to improve the performance by examining different techniques at each step of the approach using unsupervised learning methods to solve the EADP problem. The experiments showed that fluid clustering and spectral clustering yielded better results than Newman clustering and CNM clustering. Furthermore, using kernel principal component analysis or non-negative matrix factorization for feature selection improved performance compared to using all features, particularly in the case of unlabeled data. Xu, Li, et al. (2021) explored the application of clustering techniques for unsupervised EADP, but they only used F1@20%effort and *Popt* as the effort-aware metrics. They did not consider the impact of IFA, which is crucial because the unsupervised method ManualUp has not been further investigated due to its high IFA. Therefore, we compare the clustering techniques with the unsupervised method ManualUp and some supervised EADP methods to evaluate their performance under six effort-aware metrics. Our work is more comprehensive and reflects the difference in the performance of different clustering techniques for unsupervised EADP in a more comprehensive manner. Liu et al. (2022) introduced the Transfer Spectral Clustering (TSC), an unsupervised model for identifying the software defects without labeled data. TSC transfers knowledge from auxiliary unlabeled data in the source project to enhance clustering on unlabeled data in the target project. In experiments with seven software projects, TSC outperformed four unsupervised methods and showed competitive performance against eight supervised cross-project methods. Thirumoorthy and Britto (2022) introduced ESAMP-SMO, a hybrid elitist self-adaptive multi-population social mimic optimization technique for software defect module clustering. The objective function minimizes intra-cluster distance while maximizing fault prediction rate. Using NASA datasets, the proposed technique demonstrates superior performance compared to other competitor approaches in the performance comparison analysis. Khalid et al. (2023) introduced a method that employs K-means clustering for class label categorization, applies classification models to selected features, and utilizes Particle Swarm Optimization for model optimization.

3. Preliminaries

Unsupervised clustering techniques have the ability to identify defective software modules without the need for defect labels. Therefore, searching for unsupervised clustering technologies that can achieve a performance similar to or better than supervised EADP models is highly valuable. We provide a brief description of the process of using clustering techniques for unsupervised EADP in our work and offer a concise introduction to the 22 unsupervised clustering techniques. These models are classified into seven clustering families, as outlined by Xu, Li, et al. (2021).

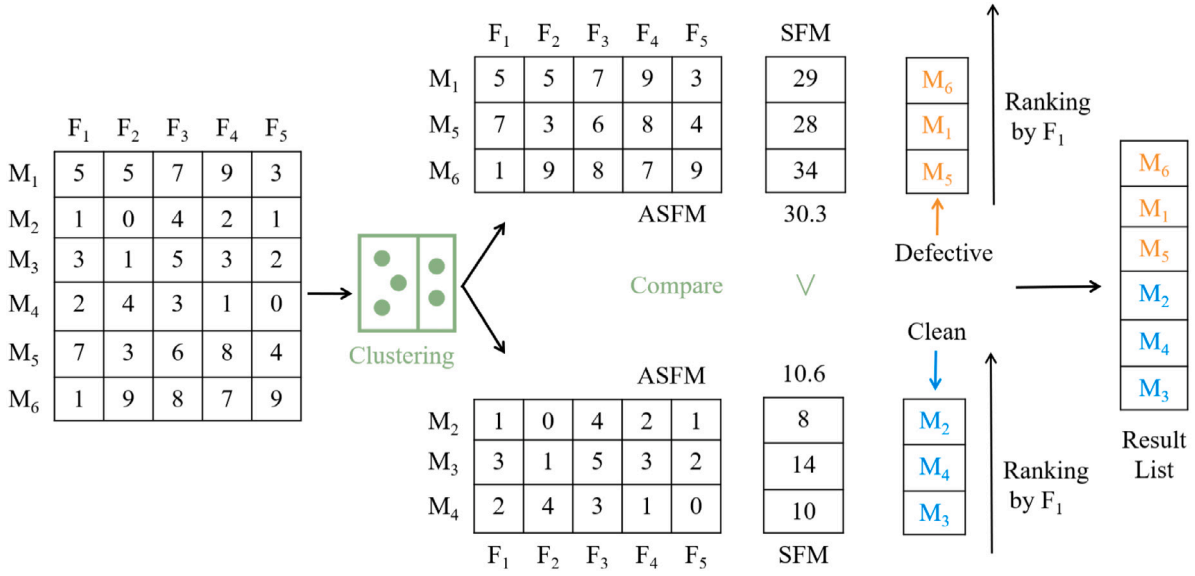

Fig. 3. The example process of using clustering techniques for unsupervised EADP.

3.1. Process of clustering techniques for unsupervised EADP

According to the research of Yang et al. (2016) and Xu, Li, et al. (2021), which suggest that the cluster with higher average feature values should be labeled as defective, we adopt the labeling scheme consistent with Xu, Li, et al. (2021).

Specifically, we apply unsupervised clustering techniques to partition the modules into two distinct clusters. Then, we proceed to compute the Sum of Feature values of each Module (SFM) in both clusters, which allows us to derive the Average SFM (ASFM) for each cluster. For binary clustering, we label the modules in the cluster with higher ASFM values as defective and those in the other cluster as clean. For multi-clustering, we label the modules in the clusters with ASFM values exceeding the mean value of these ASFMs as defective and those in the remaining clusters as clean. The above process is equally applicable to multi-clustering techniques. Next, we sort the modules in the defective and clean clusters in ascending order based on a software feature separately. This approach aligns with Menzies et al. (2010), suggesting that smaller feature values often correspond to higher defect densities, a phenomenon known as "smaller modules inspected first". Finally, we place the defective modules before the clean modules to obtain the final ranking result. The example in Fig. 3 depicts the entire process for a dataset consisting of six modules, each having five-dimensional software features. We cluster these six modules into two clusters using unsupervised clustering techniques, where cluster 1 comprises modules M_1, M_5, and M_6, while cluster 2 comprises modules M_2, M_3, and M_4. Next, we compute and compare the ASFM values of these two clusters. The ASFM value of cluster 1, which is 30.3, exceeds that of cluster 2, which is 10.6. Therefore, we label the three modules in cluster 1 as "defective" and the modules in cluster 2 as "clean". Then, suppose we sort the modules in both clusters in ascending order based on the first feature value. Finally, we arrange the defective modules before the clean ones to obtain the final ranking result (i.e., M_6, M_1, M_5, M_2, M_4, and M_3.).

3.2. Clustering techniques

3.2.1. Partition-based clustering (PBC)

The PBC methods divide the dataset into non-overlapping subsets, with each subset corresponding to a cluster, and the clusters are mutually independent. The PBC methods begin by selecting the number of clusters to be formed and initializing the cluster centroids. Next, the PBC methods calculate the distance of each module in the dataset to all the cluster centroids and assign each module to the cluster whose centroid is closest. After assigning each module to a cluster, the centroid of each cluster is re-calculated, typically by taking the mean of all modules in the cluster. This process of re-assigning modules to clusters and re-calculating the centroids is repeated until a stopping criterion is met, such as reaching the maximum number of iterations or when the centroids no longer change. Finally, each module in the dataset is assigned to a cluster, and the clusters formed by the PBC method are mutually independent and non-overlapping subsets of the dataset.

In our study, we opt for several clustering algorithms that are based on K-means (Ikotun et al., 2022) and have undergone modifications, namely K-medoids (Lund & Ma, 2021), X-means (Mughnyanti et al., 2020), Fuzzy C-Means (FCM) (Askari, 2021), G-means (Sudakov & Dmitriev, 2022), MiniBatchKmeans (Mehta et al., 2022), and K-means++ (Li & Wang, 2022).

3.2.2. Hierarchy-based clustering (HBC)

The HBC methods construct a clustering hierarchy by continually merging or splitting clusters, resulting in clustering results at different levels. For each pair of software modules, HBC methods calculate their similarity using a similarity metric such as Euclidean distance or Manhattan distance, and record all similarities in a similarity matrix. Based on the similarity matrix, HBC methods use a hierarchical clustering algorithm to gradually merge the most similar clusters until all software modules are clustered into a single cluster. By selecting an appropriate number of clusters based on the clustering tree, such as by setting a threshold or truncating the tree, the software modules can be partitioned into the chosen clusters to obtain the final clustering result (Xu, Li, et al., 2021).

In our work, we opt for four methods in HBC, including Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) (Yin et al., 2020), Clustering Using REpresentatives (CURE) (Guha et al., 2001), RObust Clustering using linKs (ROCK) (Guha et al., 2000), and Agglomerative Hierarchical Clustering (AHC) (Ding & He, 2002).

3.2.3. Density-based clustering (DBC)

The DBC methods are designed to discover clusters by identifying high-density regions within a given dataset. These methods use density connectivity to determine the boundaries of clusters and can handle clusters of arbitrary shapes. To begin, the distance threshold ϵ and

the minimum number of modules $MinPts$ are set. For each software module, its density is calculated, which is the number of software modules in its neighborhood within the distance threshold ϵ. If the density is greater than or equal to the minimum number of modules $MinPts$, the software module is marked as a core module. For each core module, all density-reachable modules are added to the same cluster. Specifically, the process begins with the core module, where software modules with a density not less than $MinPts$ within the distance threshold ϵ are added to the cluster. If the added module is also a core module, all modules in its neighborhood are added to the cluster, and this process is repeated until all density-reachable modules have been included in the cluster. The software modules not included in any cluster are marked as noise modules.

In our work, we opt for three methods in HBC, including Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Deng, 2020), Ordering Points To Identify Clustering Structure (OPTICS) (Subudhi & Panigrahi, 2022), and Mean Shift (MS) (Ranjbarzadeh & Saadi, 2020).

3.2.4. Model-based clustering (MBC)

The MBC methods describe the data generation process using a probability or other mathematical model, determine the number and shape of clusters through model parameter estimation, and discover elliptical or spherical clusters. First, the MBC methods calculate its probability of belonging to each probability model for each software module, and assign it to the most likely cluster based on the maximum a posterior probability principle. Then, based on all the software modules in the current cluster, estimate the parameters of each probability model. If there are changes in the clustering assignment of software modules, reassign the clusters; otherwise, assume that the optimal clustering result has been obtained.

In this work, we opt three MBC methods including Self-Organizing Map for Simple Clustering (SOMSC) (Peng & Nie, 2017), SYNChronized SOM (SYNC-SOM) (Novikov & Benderskaya, 2014), and Expectation Maximization (EM) (Subudhi et al., 2020).

3.2.5. Graph-theory-based clustering (GTBC)

The GTBC methods define the similarity between software modules as edge weights on a graph, discover clusters through graph connectivity, and can handle clusters of arbitrary shapes. Firstly, a similarity graph in the GTBC method is constructed between software modules. Then, based on the connectivity and cut properties of the graph, the graph is partitioned into multiple subgraphs, with each subgraph corresponding to a cluster. During implementation, it is typically necessary to set a threshold to control the size of the subgraphs, and to choose an appropriate similarity measurement method to construct the similarity graph.

In this paper, we opt for one GTBC method named Affinity Propagation (AP) (Frey & Dueck, 2007).

3.2.6. Sequence-based clustering (SBC)

The SBC methods perform clustering based on the similarity of sequence data. These methods are designed to handle data with temporal structure and are useful for discovering patterns and trends within sequences. The first step in the SBC method involves converting the sequence data into feature vector representations. Next, an appropriate similarity measure is used to compute the similarity between sequence data. Finally, the sequence data is clustered based on the similarity matrix.

In this work, we opt three SBC methods, including Basic Sequential Algorithmic Scheme (BSAS) (Theodoridis & Koutroumbas, 2006), Two-Threshold Sequential Algorithmic Scheme (TTSAS) (Ahmadi & Berangi, 2008), and Modified BSAS (MBSAS) (Angel & Viola, 2016).

3.2.7. Grid-based clustering (GBC)

The GBC methods divide the data space into multiple grids and perform clustering on each grid, enabling fast processing of large datasets and obtaining good clustering results for spatially clustered data. It involves dividing the data space into multiple grids and then, for each grid, counting the number of software modules and density information within it. Based on a density threshold, the grid is labeled as a core region, an edge region, or a noise region. Next, for all core regions, clusters are formed by building them around their centers and adding software modules from neighboring regions to the cluster.

In this paper, we opt for one GBC method named BANG (Schikuta & Erhart, 1998).

The succinct descriptions for the 22 unsupervised clustering techniques of the seven clustering families are presented in Table 1.

4. Experiment setup

4.1. Dataset

As our research requires comparison with supervised EADP methods, we employ cross-version validation while constructing the supervised EADP models. Cross-version validation enables the evaluation of a model's generalization ability across different versions of software systems. If a model performs well in multiple versions of software systems, it can be deemed as robust and reliable, and can be effectively applied in practical applications (Tantithamthavorn et al., 2018).

To meet the requirements of the EADP task for the number of defects, we select the PROMISE data repository (Boetticher, 2007) in this paper. The selection of our experimental dataset differs from previous research (Chen et al., 2018; Cheng et al., 2022; Fu & Menzies, 2017; Huang et al., 2019; Yang et al., 2021; Zhao et al., 2022) as they did not consider the information on the number of defects and chose the datasets without the number information, such as SOFTLAB (Turhan et al., 2009), RELINK (Wu et al., 2011), NASA (Shepperd et al., 2013), Change-level datasets (Kamei et al., 2012), and Android datasets (Catolino et al., 2019). However, the goal of EADP is to predict defect density (i.e., the ratio between the number of defects and LOC) and rank software modules based on the density (Yu et al., 2024). Furthermore, it is important to note that the PofB@20% metric we utilize requires information on the number of defects. A detailed explanation of this metric will be provided in Section 4.2 of the paper. Therefore, we finally select the PROMISE dataset, which is an open-source dataset comprised of 41 releases from 11 open-source software projects. This dataset has been widely utilized in EADP research (Ni, Xia, Lo, Chen, & Gu, 2022; Xu, Li, et al., 2021; Yan et al., 2017; Yu, Bennin, et al., 2019; Yu et al., 2017). More detailed information is provided in Table 2, where the column "#Module" denotes the count of modules, "#Def" corresponds to the number of defects, "%Def" signifies the percentage of defective modules, and "AvgDef" indicates the mean number of defects. The PROMISE dataset contains 20 software features that cover various aspects of software complexity, coupling, cohesion, reusability, maintainability, reliability, and scalability.

4.2. Evaluation metrics

4.2.1. Effort-aware evaluation metrics

Since the effort-aware metrics used in Xu, Li, et al. (2021) are insufficient to comprehensively evaluate model performance, we employ the following six commonly used effort-aware evaluation metrics from EADP research (Amasaki et al., 2022; Cho et al., 2022; Huang et al., 2019; Khatri & Singh, 2022; Ni, Xia, Lo, Chen, & Gu, 2022; Rao et al., 2021; Xu, Zhao, et al., 2021; Yang et al., 2020; Yu, Liu, et al., 2019) to evaluate these clustering techniques. Similar to these EADP studies, we restrict the limited effort to 20% of the total LOC of the defect dataset. Assume that there are N software modules in a defect dataset, which contain P defective modules and Q defects. When checking the top 20%

Table 1
A summary of the unsupervised clustering techniques.

Family	Method	Brief Description
PBC	K-means	A centroid-based clustering algorithm by selecting the average values of the instances in the cluster as the centers.
	K-medoids	It selects instances from the dataset as representatives, known as modoids, to build clusters around.
	X-means	It initially assumes a minimum number of clusters and dynamically increases them using a specified splitting criterion to control the process of splitting clusters.
	FCM	It allows modules to belong to multiple clusters simultaneously by assigning them membership degrees that represent their degrees of belonging to each cluster.
	G-means	It grows the number of centers by splitting those centers whose data do not appear to come from a Gaussian distribution based on a statistical test.
	MiniBatchKmeans	A variant of the K-means algorithm that updates the cluster centers using small, random subsets of the data.
	K-means++	It selecting the initial centers in K-Means, which guarantees an approximation ratio O(log k) and aims to find optimal initial centers to improve clustering results.
HBC	BIRCH	It uses Clustering Features and a CF tree to incrementally and dynamically cluster multi-dimensional metric data modules.
	CURE	It utilizes a hierarchical clustering approach, and optimizes performance using a KD-tree implementation.
	ROCK	It uses common neighbor distance to detect clusters, and iteratively removes links between distant clusters until convergence.
	AHC	A bottom-up hierarchical clustering algorithm that merges smaller clusters to form larger clusters.
DBC	DBSCAN	A density-based clustering algorithm that groups nearby instances and identifies outliers with distant neighbors.
	OPTICS	A density-based clustering algorithm that produces a hierarchical clustering with a reachability distance plot.
	MS	Iteratively shifting instances until they reach the nearest peak of their kernel density estimation surface.
MBC	SOMSC	It uses a self-organizing map to group similar instances together based on their topological relationships in a low-dimensional space.
	SYNC-SOM	A bio-inspired clustering algorithm that uses a synchronized oscillatory network based on SOM as the first layer to perform data clustering.
	EM	An iterative algorithm that alternates between an E-step and an M-step, which respectively compute the expectation of the log-likelihood and maximize the log-likelihood to update the model parameters.
GTBC	AP	It uses message passing between instances to find the most representative exemplars as cluster centers.
SBC	BSAS	It selects a single vector as the representative of each cluster and aims to create compact clusters with non-repeating vectors.
	MBSAS	A modification of the BSAS that runs through the instances twice, allowing for better cluster separation.
	TTSAS	A modification to BSAS and MBSAS that uses two threshold parameters to assign modules to clusters and create new clusters.
GBC	BANG	It uses a grid structure to organize the value space and groups patterns into blocks, which are then clustered using a topological neighbor search algorithm.

LOC according to the predicted result of the EADP model, the software testing team inspects n software modules and finds p actual defective modules with q defects. In addition, the software testing team have inspected k modules when they detect the first actual defective module.

Precision@20% is defined as the quotient of the number of truly defective modules to the number of predicted defective modules in the top 20% LOC. A lower Precision@20% could affect the testing team's confidence, which can squander valuable testing resources.

$$Precision@20\% = \frac{p}{n} \tag{1}$$

Recall@20% is the proportion of truly defective modules identified in the top 20% LOC to the total number of defective modules present in the dataset. More defective modules could be discovered with a higher value of Recall@20%.

$$Recall@20\% = \frac{p}{P} \tag{2}$$

F1@20% takes into account both the Precision@20% and Recall@20% simultaneously, which is a harmonic average of the two.

$$F1@20\% = \frac{2 \times Precision@20\% \times Recall@20\%}{Precision@20\% + Recall@20\%} \tag{3}$$

PofB@20% is the Proportion of the found Bugs when the top 20% LOC are inspected. This metric is equivalent to Recall@20% when each defective module comprises only one defect. A higher PofB@20% signifies the ability to detect a larger number of defects.

$$PofB@20\% = \frac{q}{Q} \tag{4}$$

PMI@20%[1] is the Proportion of Module Inspected when the top 20% LOC are inspected (Huang et al., 2017; Yu et al., 2023). A high value of PMI@20% implies that the software testing team needs to scrutinize a greater number of modules for the same number of LOCs. This increases the actual effort and time cost as they frequently switch between different modules.

$$PMI@20\% = \frac{n}{N} \tag{5}$$

PofB@20% and PMI@20% are often correlated, and when an EADP method obtains a high PofB@20%, it will also achieve a high PMI@20%.

IFA is the number of Initial False Alarms encountered before the testing team finds the first actual defective module. Kochhar et al. (2016) found that if the top-k predicted defective modules are all false defective modules, the software testing team may become frustrated and stop checking other modules for defects. In addition, their study shows that almost all respondents (close to 98%) agree that examining more than ten actual clean modules was beyond their acceptable level. Recent studies (Huang et al., 2019; Li, Yang, et al., 2023; Yu et al.,

[1] Initially (Huang et al., 2017) proposed PCI@20% based on code change-level, and other researchers (Li, Lu, et al., 2023; Li, Yang, et al., 2023; Yu et al., 2023, 2024) transformed the code change-level PCI@20% into the module-level PMI@20%. Our methodology and evaluation metrics are tailored to the module level, and we have chosen PMI@20% to assess the efficiency of our approach.

Table 2
The details of the experiment dataset.

Project	Version	#Module	#Def	%Def	AvgDef
Ant	1.3	125	33	16	1.65
	1.4	178	47	22.5	1.18
	1.5	293	35	10.9	1.09
	1.6	351	184	26.2	2
	1.7	745	338	22.3	2.04
Camel	1	339	14	3.8	1.08
	1.2	608	522	35.5	2.42
	1.4	872	335	16.6	2.31
	1.6	965	500	19.5	2.66
Ivy	1.1	111	233	56.8	3.7
	1.4	241	18	6.6	1.12
	2.0	352	56	11.4	1.4
Jedit	3.2	372	382	33.1	4.24
	4.0	306	226	24.5	3.01
	4.1	312	217	25.3	2.75
	4.2	367	106	13.1	2.21
	4.3	492	12	2.2	1.09
Log4j	1.0	135	61	25.2	1.79
	1.1	109	86	33.9	2.32
	1.2	205	498	92.2	2.63
Lucene	2.0	195	268	46.7	2.95
	2.2	247	414	58.3	2.88
	2.4	340	632	59.7	3.11
Poi	1.5	237	342	59.5	2.43
	2.0	314	39	11.8	1.05
	2.5	385	496	64.4	2
	3.0	442	500	63.6	1.78
Synapse	1.0	157	21	10.2	1.31
	1.1	222	99	27	1.65
	1.2	256	145	33.6	1.69
Velocity	1.4	196	210	75	1.43
	1.5	214	331	66.4	2.33
	1.6	229	190	34.1	2.44
Xalan	2.4	723	156	15.2	1.42
	2.5	803	531	48.2	1.37
	2.6	885	625	46.4	1.52
	2.7	909	1213	98.8	1.35
Xerces	1.1	162	167	47.5	2.17
	1.2	440	115	16.1	1.62
	1.3	453	193	15.2	2.8
	1.4	588	1596	74.3	3.65

2023, 2024) highlighted the significance of the IFA metric as a crucial criterion for assessing model performance. Therefore, EADP methods will not be adopted by developers when the IFA of their prediction exceeds 10.

$$IFA = k \tag{6}$$

Higher Precision@20%, Recall@20%, F1@20%, and PofB@20%, lower PMI@20% and IFA, indicating excellent EADP performance.

To better illustrate these indicators, we give an example: suppose there exists a given test dataset comprising ten software modules denoted as M_1, M_2, M_3, …, M_{10}. Among these modules, M_1, M_2, and M_3 are defective with 3, 2, and 2 bugs, respectively, while the remaining modules are non-defective. Additionally, the LOC for these ten modules sum up to 1500. Specifically, M_1, M_2, and M_3 consist of 300, 150, and 50 LOC, respectively, whereas the rest of the software modules each possess 100 LOC. Due to limited testing resources, software testers can only test the top 20% LOC, i.e., 300 LOC. There is a ranking of the software modules: M_4, M_2, M_3, M_1, M_5, M_6, M_7, M_8, M_9, M_{10}.

Fig. 4 visually depicts the detection status corresponding to this ranking. The modules to be detected during the testing of the top 20% LOC are represented by blue boxes, arranged according to their ranking order. The numbers below each module denote their respective count of defects and LOC. Owing the limitation that testers can only test the top 20% of LOC of modules, and considering that the aggregate LOC count of M_4, M_2, M_3 equates to 300, constituting 20% of the overall code lines (1500), only the first three software modules can be tested in this order.

(1) Precision@20%: We find 2 actual defective modules when we check the top 20% of LOC of modules in this ranking. Therefore, the Precision@20% of this ranking is 0.667 (=2/3).

(2) Recall@20%: The total number of defective modules for this ranking is 3. We find 2 actual defective modules when we check the top 20% of LOC of modules. Therefore, the Recall@20% of this ranking is 0.667 (=2/3).

(3) F1@20%: The Precision@20% for this ranking stands at 0.667, while the Recall@20% is also 0.667. Consequently, the F1@20% for this ranking is 0.667.

(4) PofB@20%: The total number of bugs in the test dataset is 7 (=3+2+2). Among these, the first three modules incorporate 4 bugs (=2+2). Therefore, the PofB@20% of this ranking is 0.571 (=4/7).

(5) PMI@20%: The total number of modules in the test dataset is 10. Therefore, the PMI@20% of this ranking is 0.3 (=3/10).

(6) IFA: In the given ranking, the first truly defective module is M_2. Upon the detection of M_2, testers have already finalized the assessment of one module (M_4). Therefore, the IFA of this ranking is 2.

4.2.2. Classification evaluation metrics

In Section 5, we investigate the relationship between the effort-aware performance and classification performance of clustering techniques. Therefore, we employ Precision, Recall, and F1, which are the most commonly used metrics for evaluating classification performance in the filed of software engineering and artificial intelligence (Chen et al., 2023, 2020; Li, Zou, et al., 2023; Ma, Keung, He, et al., 2023; Ma et al., 2022; Ma, Keung, Yu, et al., 2023; Yang et al., 2023; Zhang et al., 2018).

Precision measures the proportion of software modules predicted by a classification model as a certain class that actually belongs to that class. It is calculated as:

$$Precision = \frac{TP}{TP + FP}, \tag{7}$$

where TP (True Positive) represents the number of software modules correctly predicted as defective modules, and FP (False Positive) represents the number of software modules incorrectly predicted as defective modules. The higher the Precision value, the more accurate the classifier's judgment is for a given category, which implies a lower number of false defective modules.

Recall measures the proportion of actual defective modules that are correctly predicted by the model. The formula to calculate it is:

$$Recall = \frac{TP}{TP + FN}, \tag{8}$$

where FN (False Negative) represents the number of software modules incorrectly predicted as clean modules. The higher the Recall value, the stronger the classifier's ability to identify the category, and the lower the number of false clean modules.

F1 is a comprehensive evaluation metric that considers both Precision and Recall. It balances the Precision and Recall values to provide a more comprehensive evaluation of its performance. The F1 is the harmonic mean of Precision and Recall:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{9}$$

4.3. Experimental process

Our experiments employ a cross-version setup where the previous version is used as the training set, and the current version is used as the testing set. For example, in the case of the cross-version dataset composed of Ant 1.3 and Ant 1.4, the supervised EADP methods use Ant 1.3 as the training set and Ant 1.4 as the testing set. However, the unsupervised clustering techniques do not require the training set,

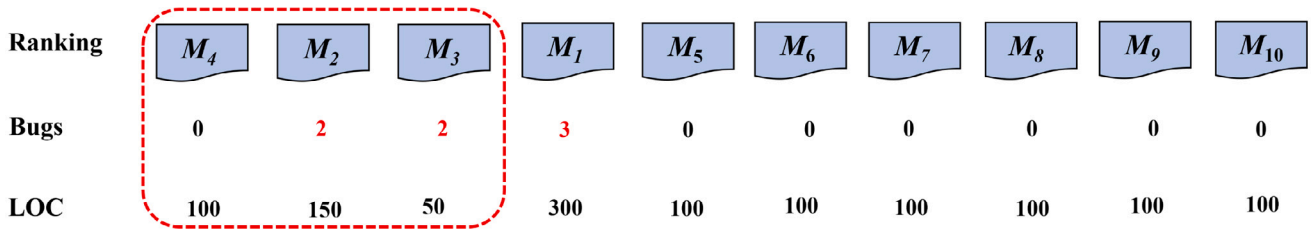

Fig. 4. An example of ranking. The modules to be detected during the testing of the top 20% LOC are represented by blue boxes, arranged according to their ranking order. The numbers below each module denote their respective count of bugs and LOC. According to this sorting, when testing resources are limited, we can only check the modules of the first 20% LOC, that is, the first 3 modules in the red box.

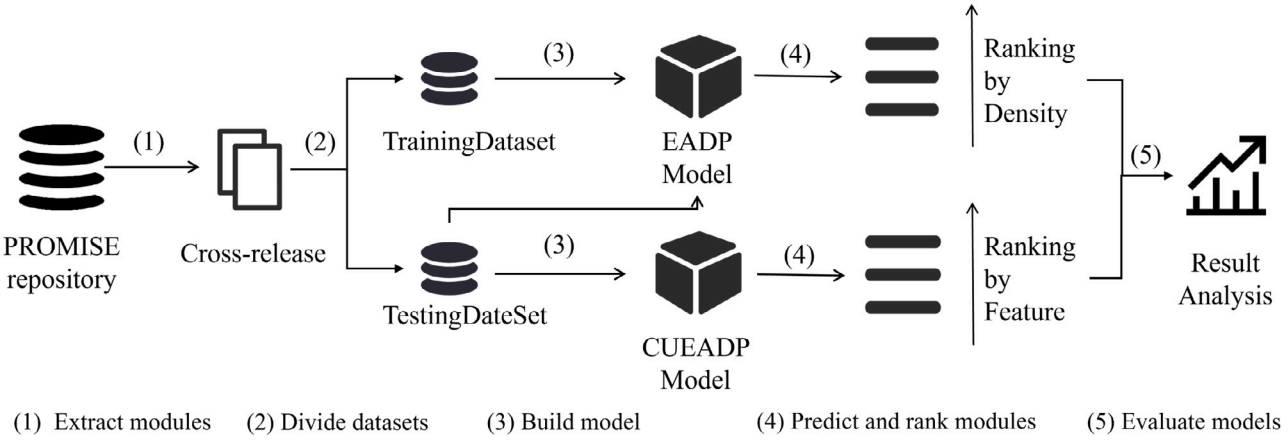

Fig. 5. The whole process of our experiment.

so these methods only utilize the Ant 1.4 dataset. Fig. 5 illustrates the complete workflow of our experiment. A detailed description of our experiment's entire process is presented as below:

We first apply a cross-version setup to split the dataset into a training dataset and a testing dataset. For the unsupervised clustering techniques, we use them to cluster the testing dataset. We subsequently employ the ASFM labeling scheme, which is described in Section 3, to categorize the clustering results into defective and clean modules. Meanwhile, we construct the EADP models using the training dataset and utilize them to predict the modules in the testing dataset as defective or clean modules.

Next, we sort the defective and clean modules in ascending order of each feature on the PROMISE dataset. We then identify the feature that yields the highest average PofB@20% value for each clustering technique. As a result, we obtain a definitive sequence of software modules, where all the defective modules precede the clean ones. Finally, we evaluate the modules within the top 20% LOC based on the sequence of software modules.

4.4. Supervised EADP methods

To verify the effectiveness of the clustering techniques for unsupervised EADP, we compare them with the following state-of-the-art supervised EADP methods.

(1) EALR (Effort-Aware Linear Regression) (Kamei et al., 2012): It assumes that there is a linear relationship between the defect density and the feature values. Accordingly, it constructs a linear regression model using the least squares method to derive parameter solutions that minimize the total difference between actual and predicted defect densities.

(2) EATT (Effort-Aware Tri-Training) (Li et al., 2020): It initially utilizes the tri-training technique to train three classifiers. Subsequently, it employs the majority voting strategy to compute the defect probability of a novel software module. Eventually, all new modules are arranged based on the ratio between the defect probability and LOC.

(3) CBS+ (Classify Before Sorting) (Huang et al., 2019): It employs a trained Logistic Regression (LR) model to identify potentially defective and clean software modules. Subsequently, the method sorts the predicted defective modules and clean modules separately based on the defect density, which is the ratio between the predicted defect probability and LOC. Lastly, the method arranges the sorted defective modules before the clean ones. Based on the research by Ni, Xia, Lo, Chen, and Gu (2022), CBS+ exhibits good performance on the JavaScript dataset.

(4) CBS+(RF): We integrate an advanced classifier known as Random Forests (RF) (Balaram & Vasundra, 2022) to replace the LR classifier in the CBS+. RF is an ensemble classifier aimed at enhancing prediction accuracy. It builds multiple decision trees by randomly selecting data subsets and features, combining their predictions through voting. This approach mitigates overfitting, enhances stability against noise, and provides insights into feature importance. The RF classifier exhibits reduced classification errors compared to existing algorithms. With its ability to handle high-dimensional data, RF is a versatile and effective method widely used for many EADP studies (Balaram & Vasundra, 2022; Pachouly et al., 2022; Zheng et al., 2022).

(5) CBS+(GB): We embed the Gradient Boosting (GB) (Sandhu & Batth, 2021) classifier into the CBS+. GB is a powerful machine learning algorithm for solving classification problems. GB operates through a three-step process. In Step 1, it optimizes the loss function. Step 2 involves predicting using a weaker learner. In Step 3, an adaptive model is fashioned by appending trees to the weaker learner, thereby minimizing the loss function. GB, serving as a remedy for the primary speed drawback of RF, is coupled with it to yield improved outcomes in EADP (Chen et al., 2022; Sandhu & Batth, 2021).

(6) SERS: We also compare a supervised method for reducing features, which we have named SERS (Shivkumar S., E James W., Ram A. and Sunghun K.) based on the authors' name of Shivaji et al. (2012). The process of this method begins by utilizing the complete feature set F and conducting feature evaluation for various feature selection

Table 3
The optimized hyper-parameters of the EADP methods. (The default parameter value is in bold font.)

Classifier	Hyper-parameters	Tuning range	Description
CBS+	tol	[0.1, 0.01, 0.001, **0.0001**, 0.00001]	Tolerance for stopping criteria
CBS+(RF)	n_estimators	[10, 20, 30, 40, 50, 60, 70, 80, 90, **100**, 120, 150]	The number of decision trees in the forest
CBS+(GB)	n_estimators	[10, 20, 30, 40, 50, 60, 70, 80, 90, **100**, 120, 150]	The number of weak classifiers

techniques. The top-performing 50% features are then selected to form a new feature set, selF. Using naive Bayes or SVM as classifiers, cross-validation is performed, and feature evaluations are recorded. The 50% of features with the lowest evaluations within selF are subsequently removed, and selF is updated accordingly. Ultimately, the best F-measure result and the corresponding feature percentage leading to the optimal outcome are determined. The overarching objective of the entire process is to iteratively select and eliminate features to identify the most optimal feature set based on the specified evaluation metric. Our experiments find that the naive Bayes can result in the best performance, aligning with the conclusions of the original paper. Therefore, we opt for the naive Bayes as the classifier for this method.

In order to attain optimal detection performance, it is imperative to fine-tune hyperparameters. Based on the outcomes of initial experiments, we utilize the grid search algorithm (Bergstra & Bengio, 2012) to optimize these hyperparameters, with the F1 score as the target for optimization. The details of the hyper-parameters of the EADP methods are shown in Table 3.

4.5. Statistic test

Scott-Knott Effect Size Difference (ESD) test is a statistical method for comparing the means of model groups based on their effect size differences at the significance level of 0.05 ($\alpha = 0.05$). It uses hierarchical clustering to partition the set of the model means into distinct groups with non-negligible differences in effect size. The method involves two steps: (1) finding a partition that maximizes model means between groups, and (2) splitting or merging groups based on the magnitude of the difference for each pair of the model means. The test aims to produce a ranking of models while ensuring that the magnitude of differences within each group is negligible, and the magnitude of differences between groups is non-negligible.

Wilcoxon signed-rank test is a non-parametric hypothesis testing method used to compare whether there is a significant difference between the medians of two independent or paired models. This method arranges the sample data, ranks them according to their order, and then calculates a statistic, the Wilcoxon signed-rank test statistic, based on the significant difference in the ranks between the two models. According to the rank-sum distribution theory in statistics, the significance level and confidence interval of this statistic can be calculated to determine whether the difference in medians between the two models is significant or not. The null hypothesis posits that there exists no difference between two distinct methods. If the p-value is less than 0.05, we reject the null hypothesis, implying a significant difference between the methods; otherwise, we shall refrain from rejecting it.

5. Experimental results

5.1. RQ1: Which unsupervised clustering techniques exhibits the most exemplary performance?

Motivation: Previous research on unsupervised defect prediction have not provided a definitive conclusion as to which unsupervised clustering technique has the most superior performance on EADP. In order to investigate which unsupervised clustering technique exhibits the most exemplary effort-aware performance on the PROMISE dataset, we have compared 22 unsupervised clustering techniques.

Methods: Figs. 6–11 illustrate the distribution of performance of different clustering unsupervised methods, employing the optimal feature ranking across all datasets, concerning Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, and IFA. In these pictures, different colors of the Scott-Knott ESD test ranking indicate different ranks and the ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'. To show the rank results more precisely, we use Figs. 12–17 to show the Scott-Knott ESD rankings of different unsupervised clustering techniques. Initially, we apply the 22 clustering techniques to cluster the testing dataset into defective and clean modules and sort the clustering results in ascending order for each feature, with all defective modules listed before the clean ones. Subsequently, we select the feature that can yield the highest average PofB@20% value among the 22 clustering techniques. Then, we evaluate the modules in the top 20% LOC, utilizing the Scott-Knott ESD test and presenting the results as a box plot to demonstrate the statistically distinct groups with significant differences. We represent the methods as X.F in these figures, where 'X' denotes the clustering-based unsupervised method and 'F' signifies the best feature by sorting the modules.

Results analysis: The K-medoids clustering technique could obtain the best overall effort-aware performance by sorting the modules in ascending order based on the LOC feature. Additionally, Kmeans++ and CURE achieve performance second only to K-medoids when using the same feature ranking the modules as K-medoids. Detailed analysis of these results are as follows:

(1) Kochhar et al. (2016) found that almost all respondents considered examining more than ten actual clean modules to be beyond their acceptable level. Therefore, our initial step is to exclude the clustering algorithms with an IFA value greater than 10. Fig. 11 presents the IFA results of these 22 clustering techniques on their optimal feature sequences. Additionally, we consider the practical implications of high PMI@20% values. Specifically, higher PMI@20% values indicate that inspecting the first 20% LOC requires checking a larger number of modules, leading to additional overhead in module switching. Fig. 10 presents the PMI@20% results of the 22 clustering techniques on their optimal feature sequences. Considering the results of IFA and PMI@20%, we can reject four clustering techniques, namely DBSCAN, OPTICS, AP, and BANG, and we will ignore them in our subsequent comparisons. In detail, both DBSCAN and OPTICS have an IFA average of 11.5, while BANG has an IFA average of 17. Furthermore, DBSCAN, OPTICS, AP, and BANG have IFA values exceeding 10 on 9, 9, 7, and 16 datasets, respectively. From Figs. 10 and 16, we can also visually observe that BANG, DBSCAN, OPTICS, and AP are at the highest level in PMI@20%, indicating that these methods exhibit the poorest performance in terms of PMI@20%. Therefore, in subsequent evaluations, we should not include these four methods since their other values no longer hold statistical significance. Excluding these four methods, the remaining 18 unsupervised clustering techniques meet the IFA requirements.

(2) After excluding the four methods of DBSCAN, OPTICS, AP, and BANG, K-medoids show the best performance in terms of PofB@20% and Recall@20%, followed by K-means++ and CURE. Specifically, K-medoids exhibits the highest PofB@20% and Recall@20% values among the remaining 18 methods, significantly surpassing all other methods by 4.84% to 8.54%. However, K-medoids has an average PMI value that is 6.72% to 12.78% higher than the other methods. CURE and K-means++ lose the best method by 8.32% and 11.32%, respectively. However, this is inevitable because PofB@20% and PMI@20% are often correlated. When an EADP method achieves a high PofB@20%, it is likely to also obtain a high PMI@20%.

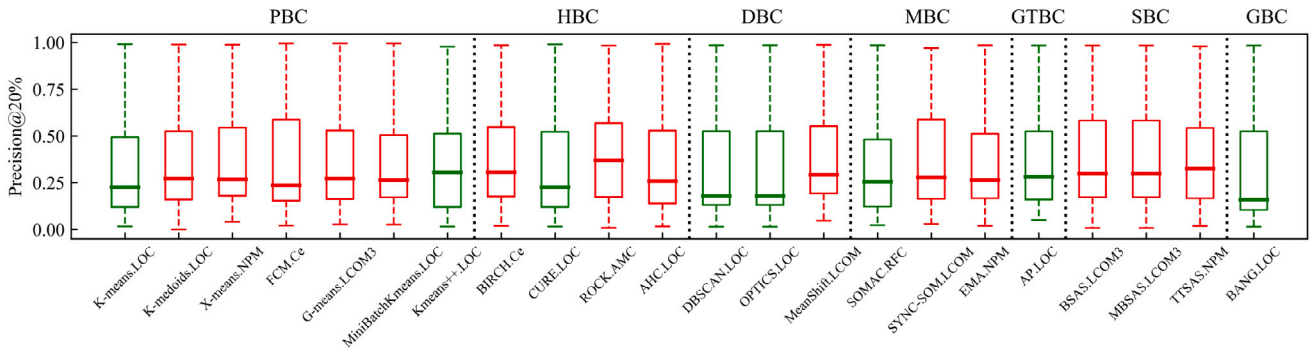

Fig. 6. The Precision@20% of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

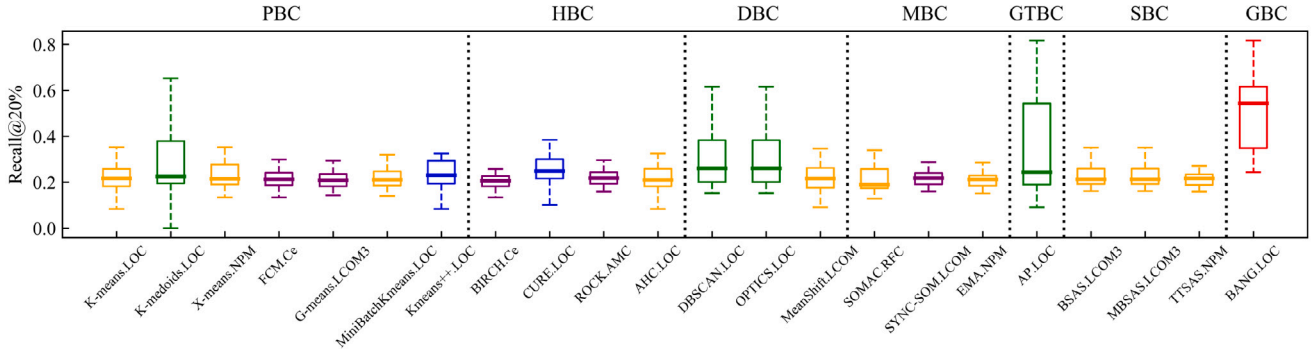

Fig. 7. The Recall@20% of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

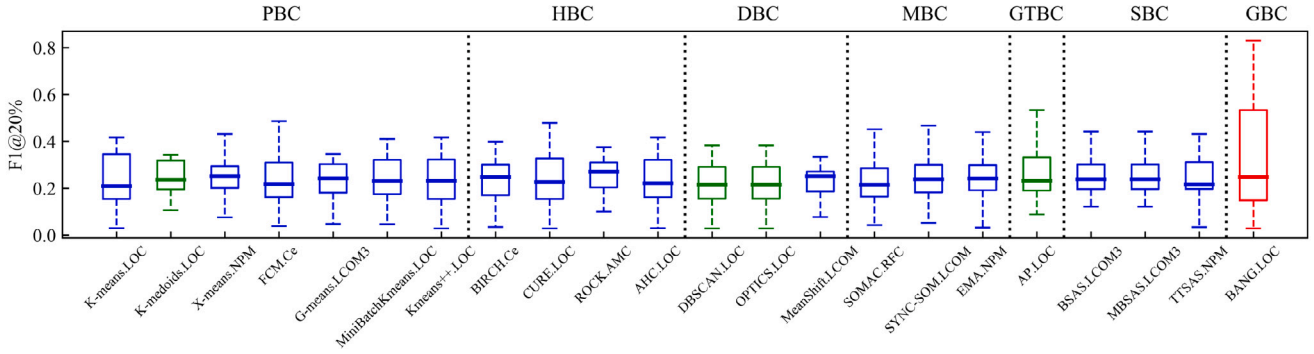

Fig. 8. The F1@20% of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

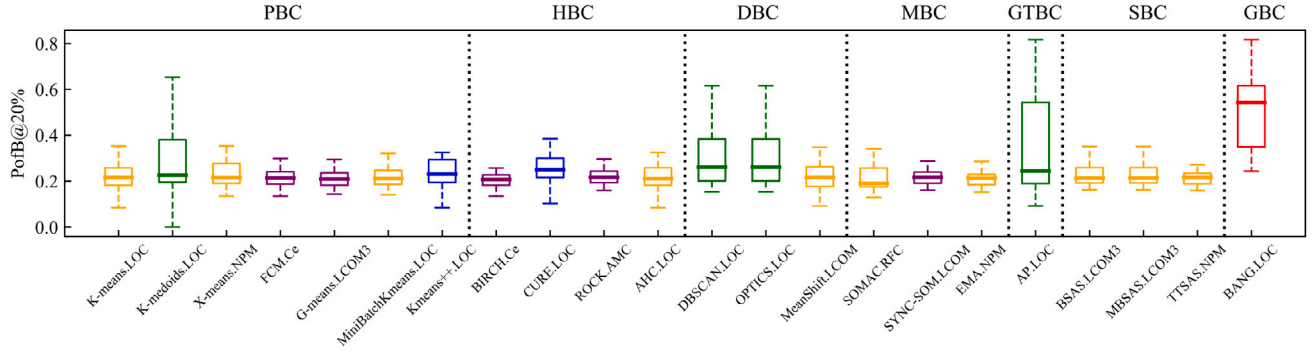

Fig. 9. The PofB@20% of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

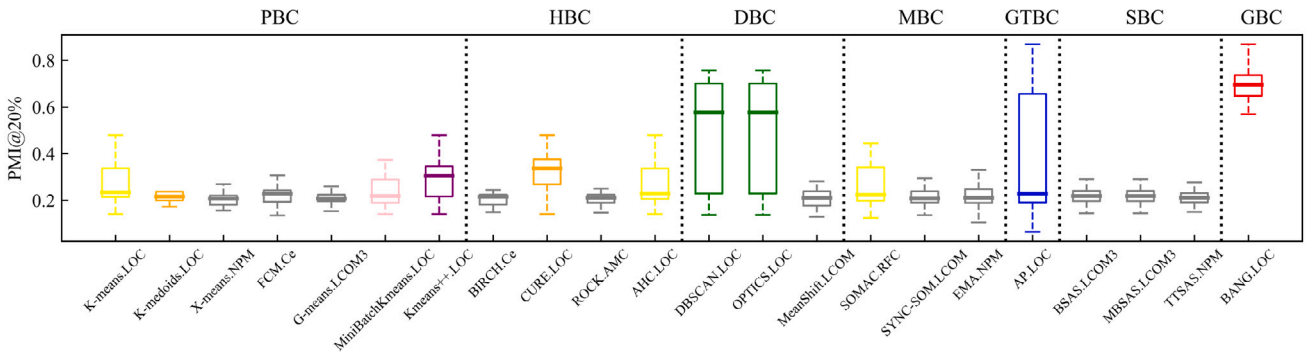

Fig. 10. The PMI@20% of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

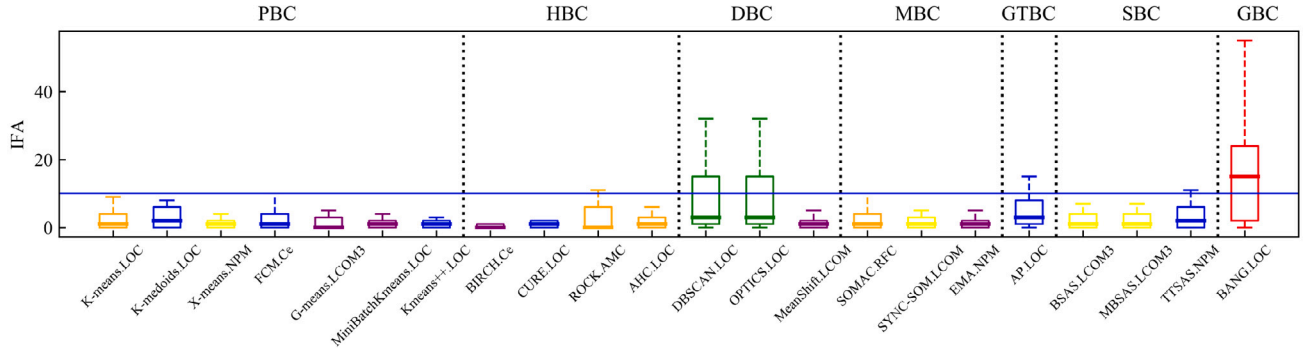

Fig. 11. The IFA of each clustering technique. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

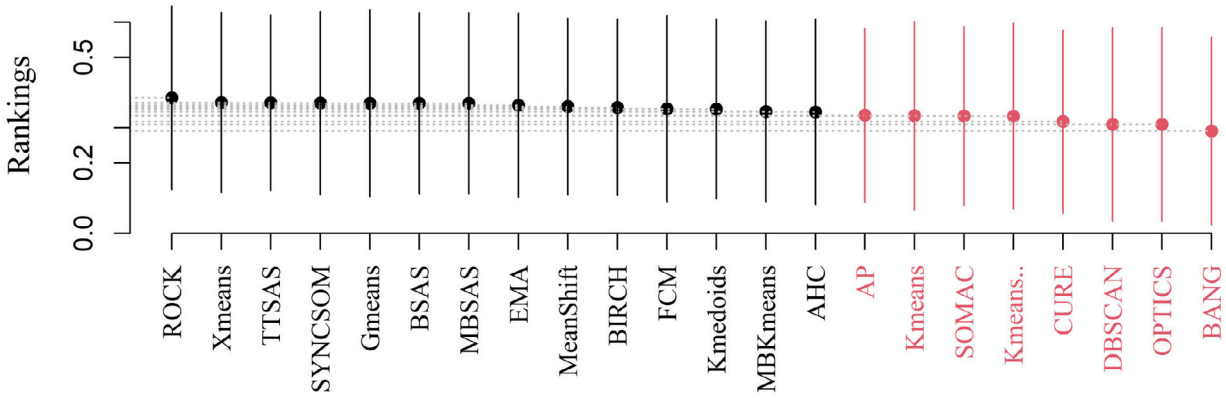

Fig. 12. The rankings of clustering techniques on Precision@20%. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on Precision@20%.

(3) The 18 methods exhibit almost identical Scott-Knott ESD levels or differ by only one level in terms of Precision@20% and F1@20%. The difference in numerical values is less than 1% on average. This indicates that all methods do not show significant differences in performance in terms of Precision@20% and F1@20%.

(4) Furthermore, we compare several algorithms selected based on the experimental results of Xu, Li, et al. (2021), which perform the best on the PROMISE dataset, including X-means, BSAS, MBSAS, DBSCAN, and OPTICS. DBSCAN and OPTICS perform particularly poorly in the IFA metric, exceeding the threshold on multiple datasets and resulting in very high PMI@20%. However, these two algorithms belong to the top-ranked group in terms of PofB@20%. Other algorithms like X-means, BSAS, and MBSAS can satisfy the IFA requirements, but no method belongs to the top-ranked group in metrics like PofB@20%, Recall@20%, etc. This implies that these clustering algorithms are more inclined to incorrectly clustering modules with small LOC as defective, leading to high IFA and PMI@20%.

> **Answer to RQ1:** K-medoids demonstrates the best overall effort-aware performance when inspecting top 20% modules, followed by K-means++ and CURE.

5.2. RQ2: Could unsupervised clustering techniques enhance the performance of the unsupervised EADP model manualup?

Motivation: In addressing RQ1, we investigate the best clustering techniques, but it remains unknown whether these methods can effectively improve the performance of ManualUp, particularly in reducing the value of IFA. Previous research has shown that ManualUp performs

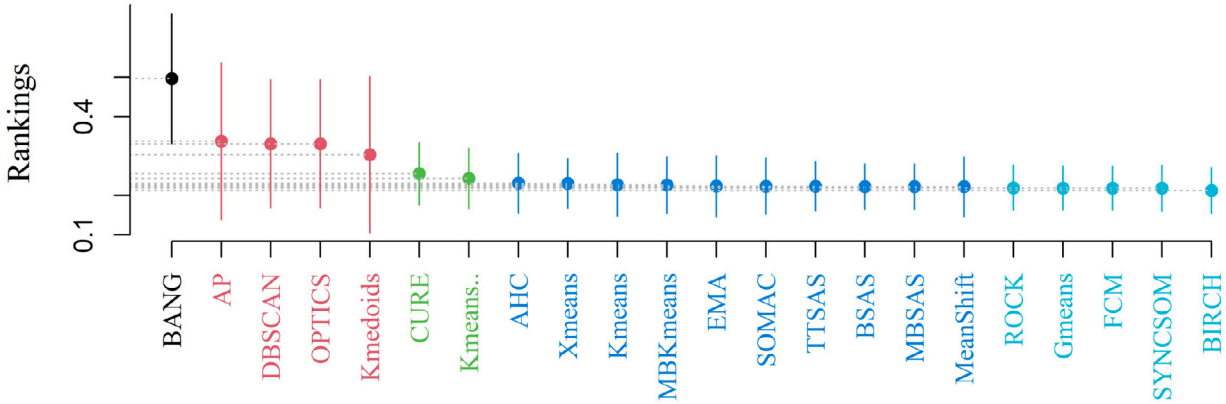

Fig. 13. The Rankings of clustering techniques on Recall@20%. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on Recall@20%.

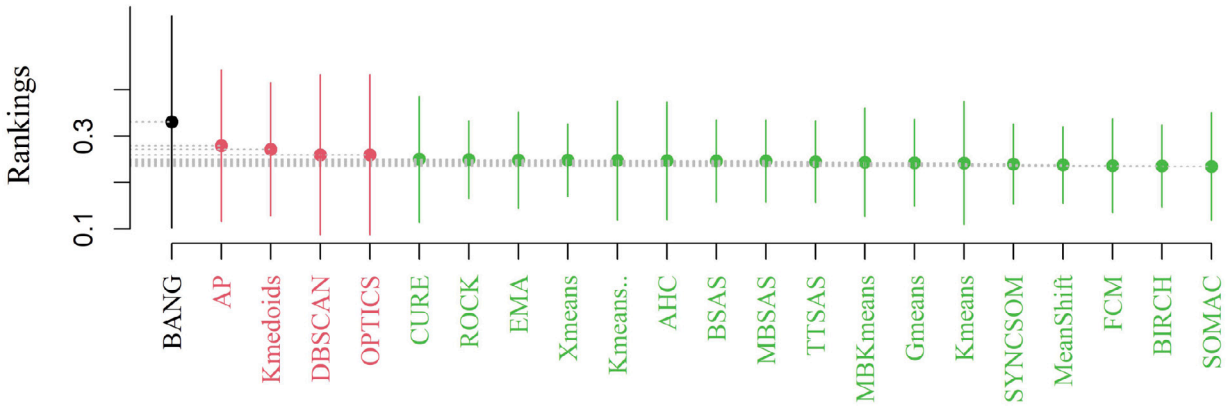

Fig. 14. The Rankings of clustering techniques on F1@20%. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on F1@20%.

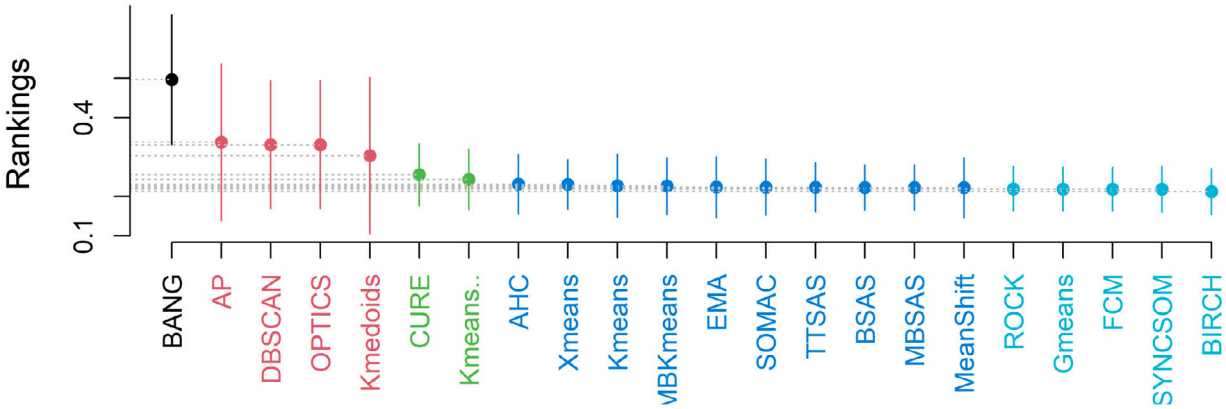

Fig. 15. The Rankings of clustering techniques on PofB@20%. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on PofB@20%.

poorly in terms of the IFA metric, with IFA exceeding 10 on multiple PROMISE datasets and even reaching a high of 66, which is considered unacceptable for the software testing team. Therefore, we conduct a comprehensive comparison between the top-performing clustering techniques (i.e., K-medoids, K-means++, and CURE) and ManualUp to explore whether unsupervised clustering techniques can improve the performance of ManualUp.

Methods: We first apply three clustering techniques divide the dataset into two clusters, and then sort the software modules based on the 20-dimensional features in ascending order according to the ManualUp approach. This is attributed to the fact that smaller feature values can exhibit a higher defect density. When inspecting the same lines of code, the ManualUp ranking strategy is capable of identifying more defective modules.

Tables 4–6 display the performance differences between these three clustering techniques and ManualUp. We also utilize the Wilcoxon signed-rank test to examine the significant differences in terms of the metrics and present the p-values at the end of Tables 4 and 5.

Results analysis: (1) The IFA values of the three clustering techniques are consistently lower than those of ManualUp across 29 cross-version validations. Particularly, the IFA values of K-medoids, K-means++, and CURE range between 6.767 and 7.433, much lower

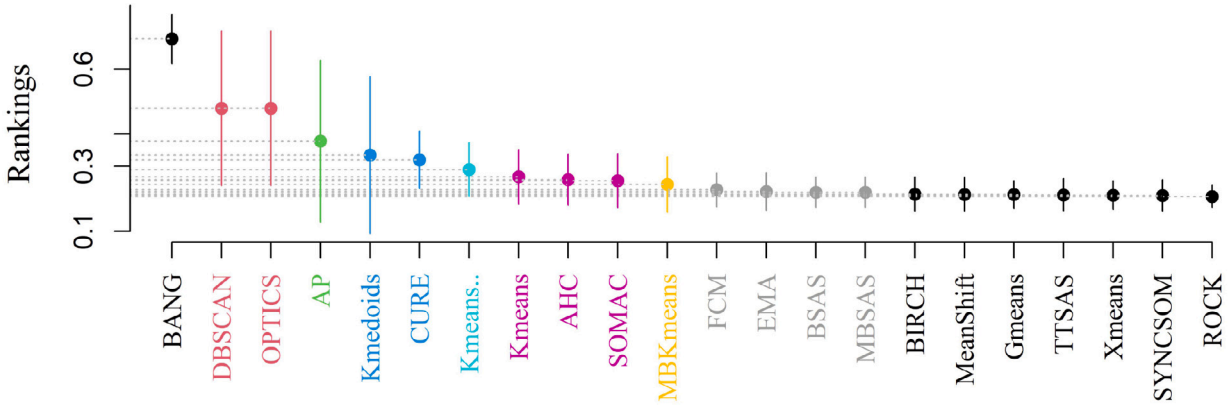

Fig. 16. The Rankings of clustering techniques on PMI@20%. Different colors indicate different ranking levels. The lower ranking value indicates the better performance of each clustering technique on PMI@20%.

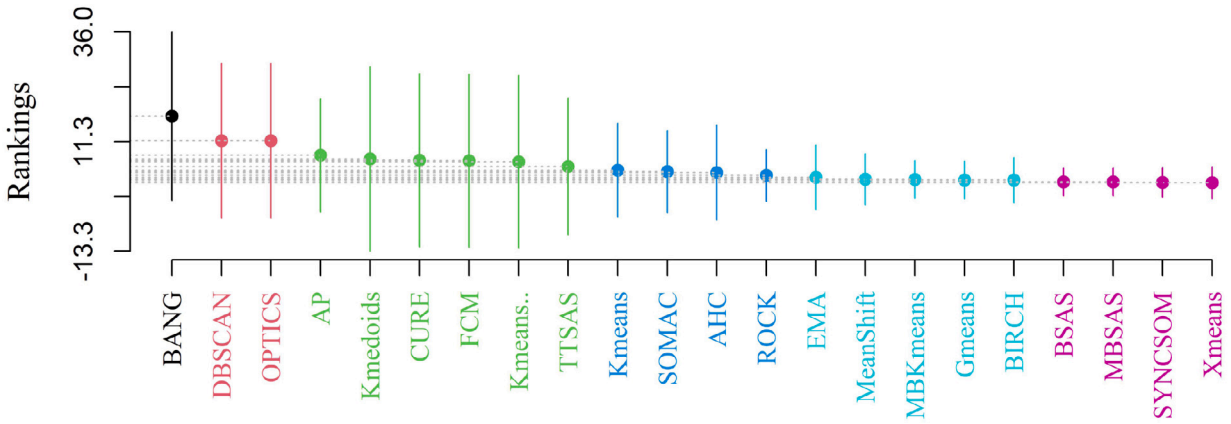

Fig. 17. The Rankings of clustering techniques on IFA. Different colors indicate different ranking levels. The lower ranking value indicates the better performance of each clustering technique on IFA.

than ManualUp's 17.033. Additionally, the p-values from the Wilcoxon signed-rank test are all less than 0.05, indicating a low probability of chance differences and significant statistical differences between the three unsupervised clustering techniques and ManualUp. Therefore, we cannot ignore this difference and must acknowledge the significant performance differences between these methods. This implies that in practical application, when testers can only utilize limited testing resources to inspect a subset of modules (i.e., modules in the first 20% LOC), introducing the best-performing clustering techniques can assist testers in identifying the first defective module earlier, contributing to building confidence among testers.

(2) Similar to IFA results, these three unsupervised clustering techniques outperform ManualUp on multiple datasets in terms of PMI@20%. K-medoids, K-means++, and CURE significantly reduce the PMI@20% by 35.9%, 40.4%, and 37.4%, respectively. This implies the introduction of the best-performing clustering techniques can significantly reduce the number of modules testes need to inspect, thereby greatly minimizing the resource wastage in switching between modules.

(3) However, on the PofB@20% and Recall@20% metrics, these three clustering techniques perform slightly worse than ManualUp. Compared to ManualUp, these three clustering techniques reduce the average value of PofB@20% and Recall@20% by 19.4%, 25.4%, and 24.2%, respectively. On the average value of Precision@20%, these three methods significantly increase by 7.2%, 4.2%, and 2.7% compared to ManualUp.

(4) In terms of the average value of F1@20%, which combines the performance of Recall@20% and Precision@20%, these three clustering techniques decrease by 6%, 8.4%, and 8.1% compared to ManualUp.

> **Answer to RQ2:** K-medoids, K-means++, and CURE can significantly reduce the IFA and PMI@20% metrics of ManualUp, without sacrificing too much performance on other effort-aware metrics.

5.3. RQ3: How does the performance of unsupervised clustering techniques compare to that of the state-of-the-art supervised EADP model?

Motivation: Due to the previous research have not explored the extent to which the unsupervised clustering technique can achieve the performance level of the supervised EADP methods, we conduct a comparative experiment similar to the ManualUp, comparing K-medoids, K-means++, and CURE with state-of-the-art EADP methods including EALR, EATT, CBS+, CBS+(RF), CBS+(GB), and SERS.

Methods: The experimental methods of K-medoids, K-means++, and CURE are consistent with RQ2. As for the supervised EADP methods, including EALR, EATT, CBS+, CBS+(RF), CBS+(GB), and SERS, they first train the models using the training dataset, and then predict on the testing dataset to obtain defect density, and rank software modules accordingly. We have tuned the main hyperparameters of CBS+, CBS+(RF), and CBS+(GB) in order to optimize their performance capabilities. The performance of these EADP methods are recorded in

Table 4
The PMI@20% value of the ten methods on each cross-version experiment when inspecting the top 20% LOC.

Cross-version	K-medoids.LOC	K-means++.LOC	CURE.LOC	ManualUp	EALR	EATT	CBS+	CBS+(RF)	CBS+(GB)	SERS
Ant1.3–1.4	0.202	0.236	0.236	0.629	0.084	0.506	**0.073**	0.096	0.096	0.112
Ant1.4–1.5	0.222	0.253	0.338	0.666	0.41	0.625	0.239	0.433	**0.137**	0.587
Ant1.5–1.6	0.199	0.217	0.447	0.655	0.177	0.573	**0.071**	0.071	0.199	0.14
Ant1.6–1.7	0.216	0.421	0.421	0.647	**0.086**	0.609	0.101	0.119	0.115	0.114
Camel1.0–1.2	0.207	0.321	0.321	0.645	0.191	0.423	0.403	**0.158**	0.505	0.176
Camel1.2–1.4	0.172	0.345	0.345	0.654	0.446	0.653	0.119	0.226	0.211	**0.092**
Camel1.4–1.6	0.199	0.316	0.316	0.66	0.382	0.617	0.326	0.106	0.096	**0.065**
Ivy1.1–1.4	**0.207**	0.344	0.344	0.743	0.39	0.714	0.357	0.34	0.32	0.423
Ivy1.4–2.0	0.173	0.384	0.384	0.736	0.531	0.716	0.628	0.193	0.534	**0.122**
Jedit3.2–4.0	**0.18**	0.376	0.376	0.745	0.376	0.68	0.232	0.252	0.245	0.229
Jedit4.0–4.1	0.734	0.356	0.356	0.734	0.391	0.718	**0.122**	0.176	0.163	0.333
Jedit4.1–4.2	0.237	0.281	0.281	0.695	0.346	0.657	**0.139**	0.161	0.161	0.174
Jedit4.2–4.3	0.23	0.268	0.268	0.703	0.181	0.6	**0.063**	0.069	0.087	0.108
Log4j1.0–1.1	0.193	0.22	0.459	0.569	0.193	0.495	**0.156**	**0.156**	0.165	**0.156**
Log4j1.1–1.2	0.215	0.478	0.478	0.605	0.351	0.58	0.156	0.161	**0.151**	0.166
Lucene2.0–2.2	0.664	**0.219**	**0.219**	0.664	0.348	0.64	0.227	0.271	0.283	0.409
Lucene2.2–2.4	0.7	**0.368**	**0.368**	0.7	0.482	0.641	0.7	0.421	0.494	0.515
Poi1.5–2.0	0.637	**0.229**	**0.229**	0.637	0.42	0.602	0.43	0.43	0.347	0.497
Poi2.0–2.5	0.208	0.179	0.179	0.639	0.278	0.519	**0.029**	0.088	0.091	0.062
Poi2.5–3.0	**0.204**	0.4	0.4	0.661	0.45	0.586	0.475	0.428	0.416	0.477
Synapse1.0–1.1	0.216	0.207	0.221	0.577	0.252	0.545	**0.072**	0.171	0.158	0.302
Synapse1.1–1.2	0.215	0.305	0.305	0.598	0.305	0.57	0.137	0.125	0.113	**0.109**
Velocity1.4–1.5	**0.22**	0.336	0.336	0.748	0.332	0.748	0.729	0.724	0.729	0.734
Velocity1.5–1.6	**0.188**	0.336	0.336	0.755	0.498	0.747	0.707	0.489	0.507	0.345
Xalan2.4–2.5	0.22	0.217	0.2	0.73	0.478	0.707	**0.056**	0.08	0.08	0.086
Xalan2.5–2.6	0.218	**0.2**	0.292	0.731	0.508	0.711	0.654	0.379	0.35	0.582
Xalan2.6–2.7	0.197	**0.194**	0.216	0.722	0.415	0.704	0.224	0.307	0.277	0.453
Xerces1.1–1.2	0.868	**0.141**	**0.141**	0.868	0.489	0.861	0.636	0.411	0.407	0.757
Xerces1.2–1.3	0.865	0.34	0.34	0.865	0.34	0.857	0.344	0.117	**0.091**	0.071
Xerces1.3–1.4	0.833	0.214	0.451	0.833	0.449	0.818	0.071	**0.068**	0.097	0.112
Average	0.335	0.290	0.320	0.694	0.350	0.647	0.289	**0.241**	0.257	0.2836
		0.593	0.453	**0.001**	0.658	**0.001**	0.275	0.109	0.170	0.271
P-value	0.593		0.025	**0.001**	**0.022**	**0.001**	0.719	**0.098**	0.299	0.614
	0.453	0.025		**0.001**	0.212	**0.001**	0.478	**0.022**	0.072	0.237

the last columns of Tables 4–6. Consistent with RQ2, we employ the Wilcoxon signed-rank test to examine significant differences in metrics, and present the results at the end of the Tables 4 and 5.

Results analysis: (1) CBS+(RF) and CBS+(GB) are the most effective methods, with the lowest IFA average value of 4.6, followed by CBS+ with 5.067. However, the IFA values of EALR and EATT are significantly higher than that of CBS+, reaching 8.733 and 17.533, respectively. Compared to the EALR method, K-medoids, K-means++, and CURE reduce the IFA value by 1.3, 1.966, and 1.6, respectively. Compared to the EATT method, K-medoids, K-means++, and CURE significantly reduce the IFA values by 10.1, 10.766, and 10.4, respectively. The IFA performance for K-medoids, K-means++, and CURE are comparable to that of CBS+. This result indicates that unsupervised clustering techniques can effectively reduce IFA and even outperform some supervised EADP models to achieve results comparable to the best supervised EADP model.

(2) As shown in Table 4, on the PMI@20% metric, K-medoids, K-means++, and CURE reduce the value by 1.5%, 6.0%, and 3.0% respectively compared to EALR and by 31.2%, 35.7%, and 32.7% respectively compared to EATT. There is a significant difference between the three unsupervised clustering techniques and EATT regarding PMI@20%. Compared to CBS+, these unsupervised clustering techniques increase the value by 4.6%, 0.01%, and 3.1%, respectively. But the differences between K-medoids, K-mean++, CBS+, and SERS are not significant. This indicates that unsupervised clustering can outperform some EADP methods on the PMI@20% metric, meaning that it can significantly reduce the number of modules that testers need to inspect for the same number of LOCs, thereby saving the cost of switching modules. Furthermore, when contrasted with the optimal EADP method CBS+(RF), these three unsupervised clustering techniques increase the average value of PMI@20% by 9.4%, 4.9%, and 7.9%. There is still a little gap between these unsupervised clustering techniques and the best EADP method.

(3) Table 5 shows that the best-performing unsupervised clustering technique K-medoids increases by 1.3% compared to EALR but decreases by 8.1% compared to EATT on the average value of the PofB@20% metric. K-medoids demonstrates a mere 0.04% deviation from CBS+ in the PofB@20% metric, indicating an almost negligible difference between the two methods. The other two unsupervised clustering techniques perform slightly worse than the six EADP methods. In comparison to the six EADP methods, these three unsupervised clustering techniques lack a distinct advantage, particularly when EATT stands out with its superior performance in the PofB@20% metric. However, the three unsupervised clustering techniques perform better on Precision@20% than EALR and EATT according to Table 6. In terms of the overall F1@20% metric, the three unsupervised clustering techniques bring about an increase in the average value, spanning a range from 8.7% to 11.4%. The best-unsupervised clustering technique K-medoids is 11.4% higher than the worst EADP method but 2.8% lower than the best EADP method.

(4) These results all indicate that unsupervised clustering techniques perform much better than some supervised EADP methods on certain metrics such as IFA, PMI@20%, and F1@20% while not always performing worse than the best-performing EADP method. For example, although unsupervised clustering techniques are slightly worse than EATT on PofB@20% and Recall@20%, they are much better than EATT on IFA and PMI@20%. This reflects the effectiveness of the unsupervised clustering approaches.

Table 5
The PofB@20% value of the ten methods on each cross-version experiment when inspecting the top 20% LOC.

Cross-version	K-medoids.LOC	K-means++.LOC	CURE.LOC	ManualUp	EALR	EATT	CBS+	CBS+(RF)	CBS+(GB)	SERS
Ant1.3–1.4	0.171	0.308	0.308	**0.525**	0.043	0.441	0.125	0.150	0.125	0.225
Ant1.4–1.5	0.172	0.212	0.273	0.312	**0.343**	0.254	0.281	0.281	0.219	0.312
Ant1.5–1.6	0.213	0.187	0.228	**0.261**	0.141	0.161	0.185	0.152	0.163	0.25
Ant1.6–1.7	0.189	0.236	0.236	0.277	**0.281**	0.186	0.253	0.247	0.253	0.265
Camel1.0–1.2	0.225	0.306	0.306	**0.602**	0.255	0.357	0.394	0.190	0.565	0.204
Camel1.2–1.4	0.195	0.248	0.248	0.469	0.301	0.326	0.255	**0.497**	0.462	0.234
Camel1.4–1.6	0.194	0.202	0.202	**0.543**	0.306	0.366	0.309	0.261	0.239	0.122
Ivy1.1–1.4	**0.400**	0.188	0.188	0.312	0.222	0.306	0.312	0.312	0.312	0.375
Ivy1.4–2.0	0.243	**0.300**	**0.300**	0.250	0.250	0.209	0.125	0.175	0.200	0.125
Jedit3.2–4.0	0.256	0.293	0.293	0.427	0.292	0.264	0.387	**0.453**	0.440	0.387
Jedit4.0–4.1	0.380	0.101	0.101	0.380	0.313	0.302	0.354	0.405	0.380	**0.418**
Jedit4.1–4.2	0.268	0.083	0.083	0.271	0.226	0.231	**0.417**	0.354	0.375	0.312
Jedit4.2–4.3	0.000	0.182	0.182	0.455	0.167	**0.508**	0.364	0.364	0.273	0.364
Log4j1.0–1.1	0.176	0.229	0.243	0.243	**0.453**	0.297	0.324	0.324	0.351	0.378
Log4j1.1–1.2	0.206	0.481	0.481	**0.614**	0.319	0.475	0.159	0.159	0.159	0.175
Lucene2.0–2.2	**0.597**	0.243	0.243	**0.597**	0.309	0.364	0.292	0.340	0.347	0.417
Lucene2.2–2.4	**0.616**	0.325	0.325	**0.616**	0.373	0.479	**0.616**	0.433	0.493	0.419
Poi1.5–2.0	**0.459**	0.270	0.270	**0.459**	0.359	0.301	0.216	0.270	0.270	0.378
Poi2.0–2.5	0.226	0.145	0.145	**0.569**	0.143	0.318	0.040	0.101	0.101	0.073
Poi2.5–3.0	0.207	0.291	0.291	**0.544**	0.306	0.387	0.484	0.456	0.438	0.509
Synapse1.0–1.1	0.210	0.226	0.250	**0.383**	0.202	0.307	0.167	0.200	0.150	0.333
Synapse1.1–1.2	0.175	0.310	0.310	**0.349**	0.303	0.274	0.198	0.198	0.186	0.198
Velocity1.4–1.5	0.229	0.312	0.312	0.683	0.181	0.503	0.683	**0.690**	**0.690**	0.683
Velocity1.5–1.6	0.182	0.321	0.321	**0.615**	0.284	0.459	0.603	0.551	0.538	0.41
Xalan2.4–2.5	0.230	0.231	0.216	**0.628**	0.405	0.541	0.096	0.109	0.106	0.119
Xalan2.5–2.6	0.225	0.207	0.242	**0.545**	0.307	0.469	0.523	0.438	0.409	0.479
Xalan2.6–2.7	0.197	0.192	0.216	**0.718**	0.378	0.620	0.227	0.311	0.281	0.455
Xerces1.1–1.2	**0.817**	0.194	0.194	**0.817**	0.591	0.787	0.606	0.662	0.620	0.718
Xerces1.2–1.3	**0.652**	0.258	0.258	**0.652**	0.326	0.463	0.116	0.159	0.159	0.13
Xerces1.3–1.4	**0.787**	0.222	0.384	**0.787**	0.327	0.464	0.094	0.092	0.128	0.146
Average	0.303	0.243	0.255	**0.497**	0.290	0.381	0.307	0.311	0.314	0.320
		0.600	0.781	**0.001**	0.967	0.052	0.738	0.719	0.658	0.393
P-value	0.600		0.025	**0.001**	**0.041**	**0.001**	0.125	0.068	**0.038**	**0.035**
	0.781	0.025		**0.001**	0.123	**0.001**	0.192	0.122	**0.080**	**0.075**

Table 6
The average Precision@20%, Recall@20%, F1@20%, and IFA values of the ten methods.

Cross-version	K-medoids.LOC	K-means++.LOC	CURE.LOC	ManualUp	EALR	EATT	CBS+	CBS+(RF)	CBS+(GB)	SERS
Precision@20%	0.353	0.333	0.318	0.291	0.326	0.299	**0.477**	0.476	0.462	0.456
Recall@20%	0.303	0.243	0.255	**0.497**	0.298	0.480	0.307	0.311	0.314	0.320
F1@20%	0.271	0.247	0.250	**0.331**	0.157	0.184	0.299	0.320	0.320	0.314
IFA	7.433	6.767	7.133	17.033	8.733	17.533	5.067	**4.600**	**4.600**	5.233

Answer to RQ3: Unsupervised clustering techniques perform better than some supervised EADP methods, especially on metrics such as IFA and PMI@20%. Although there is still a gap compared to the best EADP methods on some metrics, unsupervised clustering techniques do not always perform worse.

5.4. RQ4: Is the effort-aware performance of unsupervised clustering techniques related to the classification performance?

Motivations: As we initially employ the K-medoids to divide the modules into defective and clean ones, and subsequently rank the software modules by each feature, we ponder over whether superior classification performance can lead to better effort-aware performance. In our study, we employ three classification evaluation metrics (i.e., Precision, Recall, and F1) and six effort-aware evaluation metrics (i.e., Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, and IFA). Therefore, we undertake a correlation analysis to examine the connection between the three classification performances and the six effort-aware performances.

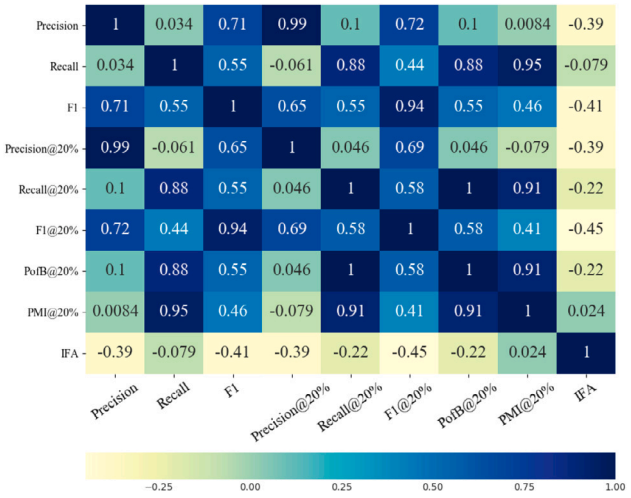

Fig. 18. The correlation among all performance measures on K-medoids.

Methods: We employ the Kendell correlation coefficient (denoted as r) to assess the correlation among the evaluation metrics and represent the results using a heatmap, depicted in Fig. 18. The intensity of the color indicates the strength of the correlation between any two evaluation metrics. According to Li, Lu, et al. (2023), we consider the correlation coefficient to be negligible ($|r| < 0.3$), low ($0.3 \leq |r| < 0.5$), moderate ($0.5 \leq |r| < 0.7$), high ($0.7 \leq |r| < 0.9$), or very high ($0.9 \leq |r| \leq 1$).

Results analysis: The Precision metric exhibits a very high correlation with Precision@20% (0.99) for K-medoids. Precision also has a high correlation with F1@20% (0.72) for K-medoids. Regarding Recall, it shows a high correlation with Recall@20% (0.88) for K-medoids. Additionally, Recall shows a high correlation with PofB@20% (0.88) and a very high correlation with PMI@20% (0.95) for K-medoids. F1 demonstrates a low correlation with PMI@20% (0.46), a moderate correlation with Precision@20% (0.65), Recall@20% (0.55), and PofB@20% (0.55), and a very high correlation with F1@20% (0.94) for K-medoids. The strong correlation between classification metrics and effort-aware metrics highlights the evident interaction between clustering quality and defect prediction accuracy. This emphasizes the importance of achieving high-quality classifying under the specified 20% effort for accurate defect prediction.

Answer to RQ4: The better comprehensive classification performance of the clustering techniques can bring better effort-aware performance to some extent.

5.5. RQ5: Can the best-performing unsupervised clustering method be generalized to other datasets?

Motivations: To explore whether the K-medoids method, which has the best performance on the PROMISE dataset, is applicable to other software defect datasets, we apply the 22 unsupervised clustering methods from RQ1 to the 16 software projects in the NASA and SOFTLAB datasets.

Methods: NASA is a widely used dataset in the field of software engineering. We utilize the clean version collected and curated by Shepperd et al. (2013), which excludes duplicated and inconsistent instances. Each selected dataset represents a NASA software system, encompassing various metrics closely related to software quality. The SOFTLAB dataset originates from a Turkish software company specializing in the development of embedded controllers for home appliances (Jing et al., 2015). The SOFTLAB dataset contains five projects (i.e., ar1, ar3, ar4, ar5, and ar6), and each project's data consists of 29 static code features.

Table 7 illustrates the number of modules (#Module), defective modules (#Def), and the percentage of defective modules relative to the total modules (%Def) for each project. In line with the experimental setup of RQ1, we conduct experiments comparing 22 unsupervised clustering algorithms on the NASA and SOFTLAB datasets. We use five effort-aware metrics, including Precision@20%, F1@20%, PofB@20%, PMI@20%, and IFA for evaluation. Recall@20% is omitted due to its equivalence to PofB@20% in the absence of defect quantity information. Figs. 19 to 23 depict the performance distribution of different unsupervised clustering methods. Figs. 24 to 28 display the Scott-Knott ESD rankings.

Results analysis: (1) K-medoids exhibits relatively low values on IFA and PMI@20%. As seen in Figs. 22 and 23, K-medoids has an average value close to 0.2 for PMI@20% and an average value around 4 for IFA, both at low levels (lower values are preferable for the two metrics). Figs. 27 and 28 show K-medoids' Scott-Knott ESD rankings as 15 for PMI@20% and as 8 for IFA, indicating its relatively good performance among the 22 unsupervised clustering methods.

(2) K-medoids consistently demonstrates superior performance in PofB@20%, Precision@20%, and F1@20%. Figs. 24 and 25 clearly show that K-medoids consistently ranks at the top, signifying its best performance in Precision@20% and F1@20%. Additionally, from Figs. 21 and 26, in the Scott-Knott ESD ranking for PofB@20%, K-medoids is either equal to or outperforms all other unsupervised clustering methods, followed by BANG.

Table 7
The details of the NASA and SOFTLAB datasets.

Dataset	Project	#Module	#Def	%Def
NASA	CM1	327	42	12.8
	JM1	7720	1612	20.8
	KC1	1162	294	25.2
	KC3	194	36	18.5
	MC2	124	44	35.4
	MW1	250	25	10.0
	PC1	679	55	8.1
	PC2	722	16	2.2
	PC3	1053	130	12.3
	PC4	1270	176	13.8
	PC5	1694	458	27.0
SOFTLAB	ar1	121	9	7.4
	ar3	63	8	12.7
	ar4	107	20	18.7
	ar5	36	8	22.2
	ar6	102	15	14.7

Answer to RQ5: The best-performing unsupervised clustering method on the PROMISE dataset, K-medoids, is applicable to NASA and SOFTLAB datasets and exhibits quite good predictive performance compared to other unsupervised clustering methods.

6. Threats to validity

(1) Our findings are based on 41 releases from 11 open-source software projects of the PROMISE dataset collected by Jureczko and Madeyski (2010), which have been used and validated by numerous EADP studies (Chen et al., 2018; Ni, Xia, Lo, Chen, & Gu, 2022; Yan et al., 2017). We have not selected those datasets previous studies used without the number information, such as SOFTLAB (Turhan et al., 2009), RELINK (Wu et al., 2011), NASA (Shepperd et al., 2013), Change-level datasets (Kamei et al., 2012), and Android datasets (Catolino et al., 2019), since the goal of EADP is to predict defect density (i.e., the ratio between the number of defects and LOC) and rank software modules based on the density (Yu et al., 2024). Therefore, we only utilize the PROMISE dataset, which comprises information on the number of defects as our experimental dataset. The PROMISE dataset is one of the datasets utilized by Xu, Li, et al. (2021) in their experiment, which belongs to different application fields and has different numbers of modules with different levels of defective proportions. However, many other software projects in other fields with other characteristics or programming languages are not used in our work. In addition, all software projects used in our work are developed by the open-source community, and it is not clear whether our conclusions can apply to commercial projects. In future work, we will further reduce the threat by analyzing more modules from other defect datasets.

(2) We employ 22 clustering techniques in our empirical study, because they have been widely investigated in previous EADP studies and can be easily invoked using Python library functions. In addition, we acknowledge the existence of some other unsupervised clustering techniques. The adoption of the unused methods in our work is left for future work.

(3) We employ not only widely used Precision, Recall, and F1 metrics in the classification scenario (Huang et al., 2018; Xia et al., 2016), but also Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, and IFA in the effort-aware scenario (Huang et al., 2019;

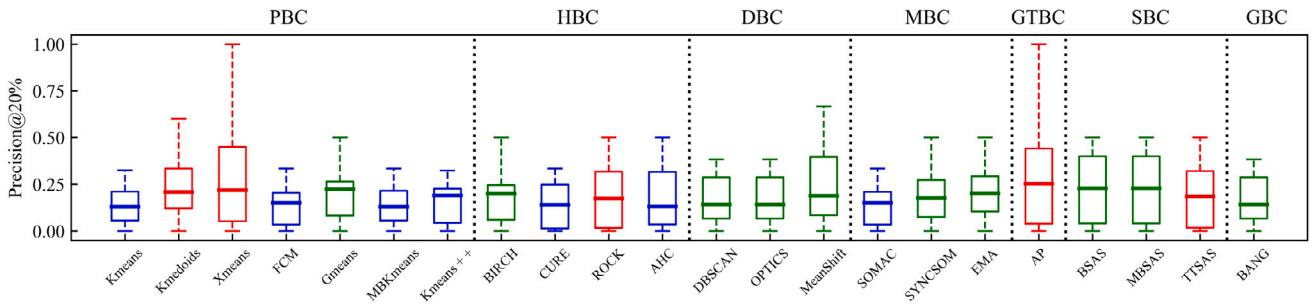

Fig. 19. The Precision@20% of each clustering technique on NASA and SOFTLAB datasets. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

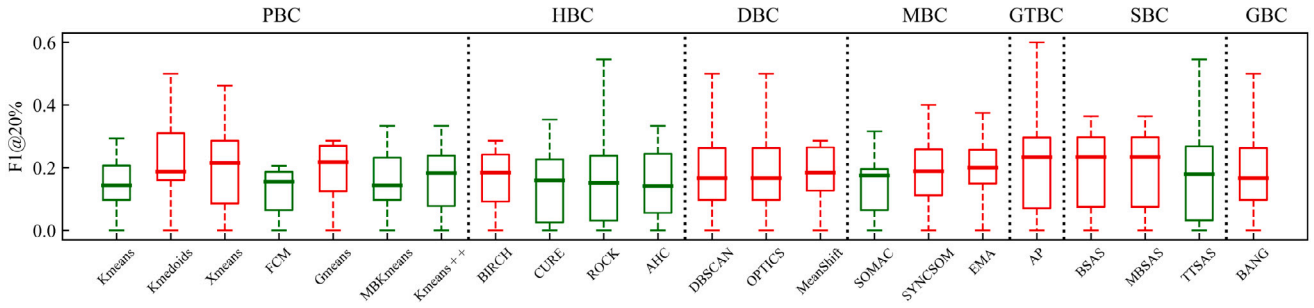

Fig. 20. The F1@20% of each clustering technique on NASA and SOFTLAB datasets. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

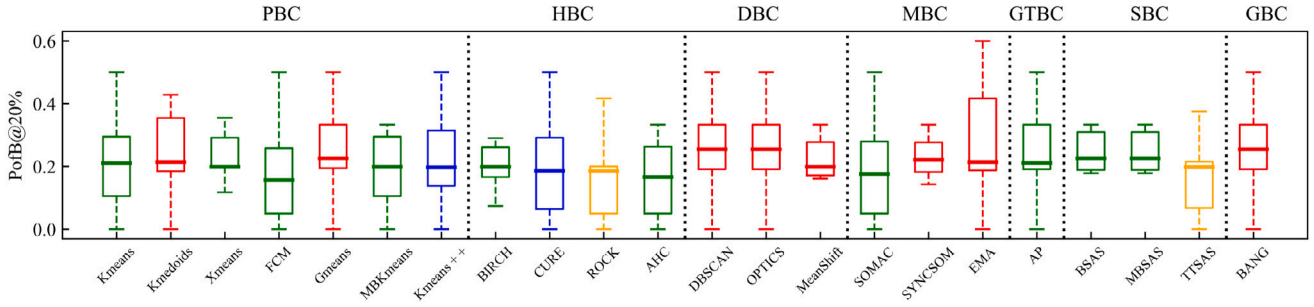

Fig. 21. The PofB@20% of each clustering technique on NASA and SOFTLAB datasets. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

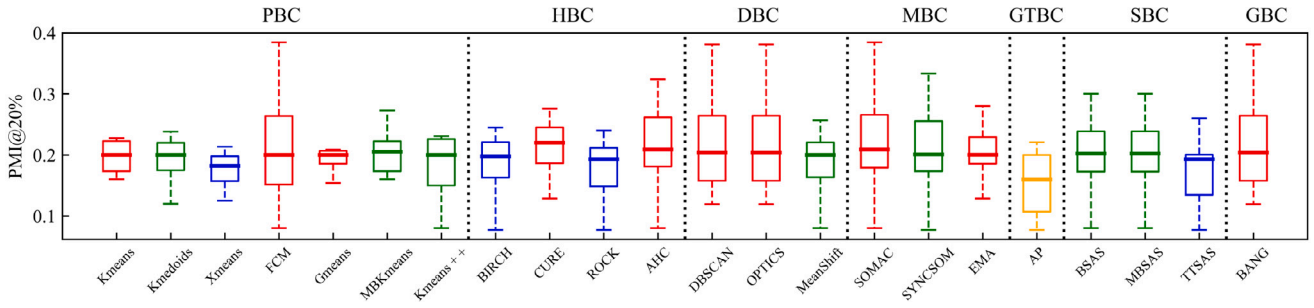

Fig. 22. The PMI@20% of each clustering technique on NASA and SOFTLAB datasets. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

Ni, Xia, Lo, Chen, & Gu, 2022; Yu et al., 2023). Since the EADP model is designed to find more defects and defective modules, we use PofB@20% and Recall@20%. We use Precision@20%, because Precision@20% and Recall@20% are usually paired. F1@20% balances the tradeoff between Precision@20% and Recall@20%, so we use F1@20% to correct the bias that may result from using Precision@20% and Recall@20%. We use PMI@20% because checking too many modules introduces additional effort costs. In addition, we use IFA because previous studies (Huang et al., 2019; Kochhar et al., 2016; Li, Lu, et al., 2023; Li, Yang, et al., 2023) have concluded that if the IFA value is too large, it will severely reduce software testers' confidence. In addition, we use the non-parametric statistical Wilcoxon signed-rank test and the Scott-Knott ESD test to compare the performance of the different clustering techniques to ensure that the differences are statistically significant.

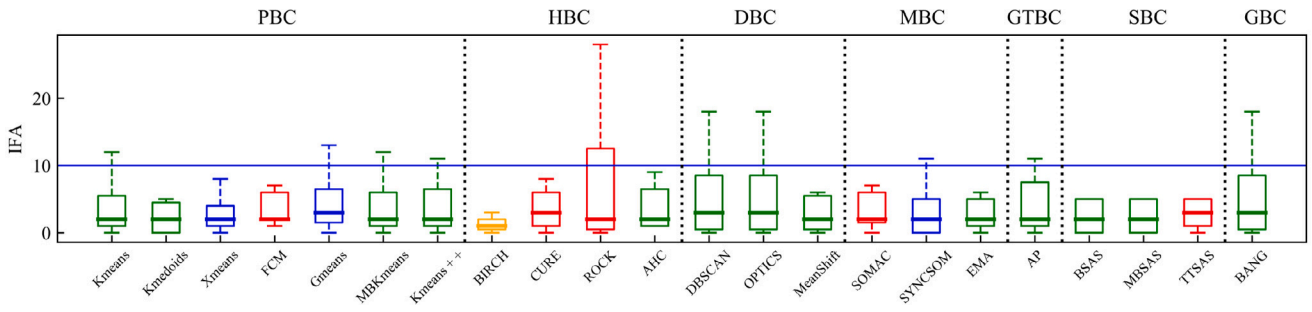

Fig. 23. The IFA of each clustering technique on NASA and SOFTLAB datasets. The ranks corresponding to the colors in descending order: 1: 'red', 2: "green", 3: 'blue', 4: 'orange', 5: 'purple', 6: 'yellow', 7: 'pink', 8: 'gray'.

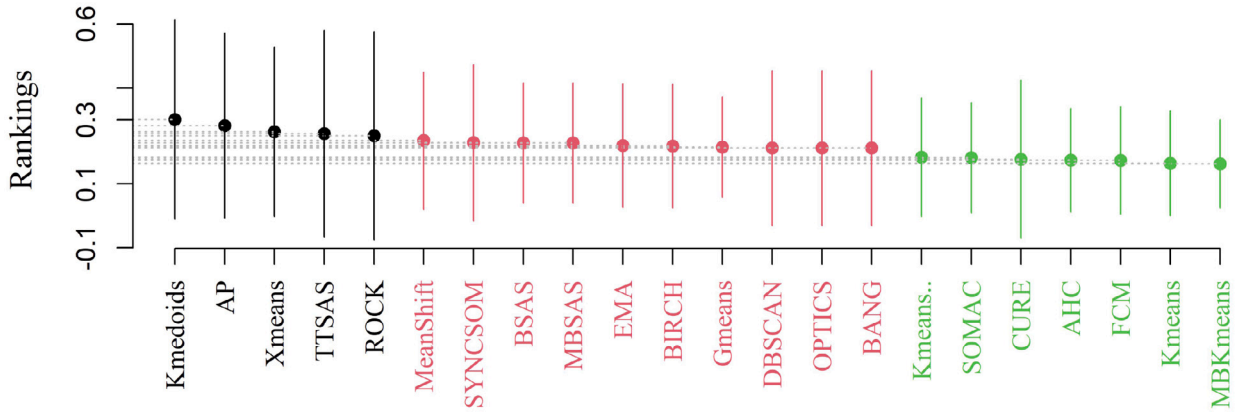

Fig. 24. The rankings of clustering techniques in terms of Precision@20% on NASA and SOFTLAB datasets. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on Precision@20%.

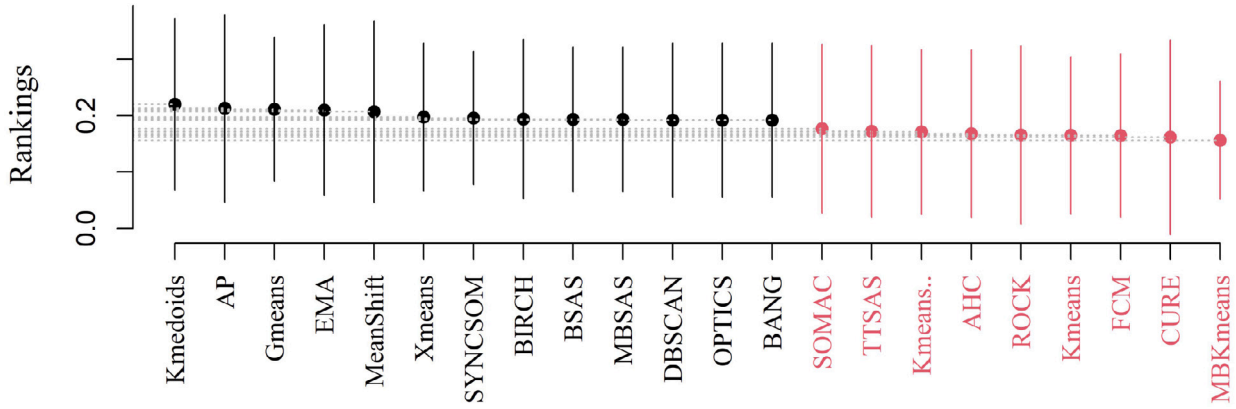

Fig. 25. The Rankings of clustering techniques in terms of F1@20% on NASA and SOFTLAB datasets. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on F1@20%.

(4) In order to alleviate the technical defects in our experiments as much as possible, we implement the classifiers based on the SKlearn library[2] and Pyclustering library.[3] We carefully implement the code for EADP methods (i.e., EALR, EATT, and CBS+) by strictly following the original papers' descriptions.

(5) The projects from the PROMISE dataset used in our paper have 20-dimensional features, each outcome generated by the clustering technologies can be sorted based on each feature. Consequently, each technology could potentially yield 20 distinct sorting possibilities. However, due to constraints on space, it is impractical to list all outcomes. Therefore, we have chosen the feature (i.e., LOC) that leads to the best results. Whether other features might result in superior outcomes for different projects remains uncertain. When it comes to sorting new projects, it is possible that the LOC might not be the best sorting criterion, but we lack an automated way to determine the ideal feature. We leave the feature selection process as a future work.

7. Implications

We present here some practical implications that researchers can draw from our experimental outcomes.

(1) **By utilizing the best-performing unsupervised clustering algorithms, testing teams can optimize resource allocation without relying on labels.** Software testers can use the three best-performing

[2] https://github.com/scikit-learn
[3] https://pyclustering.github.io/docs/0.10.1/html/index.html

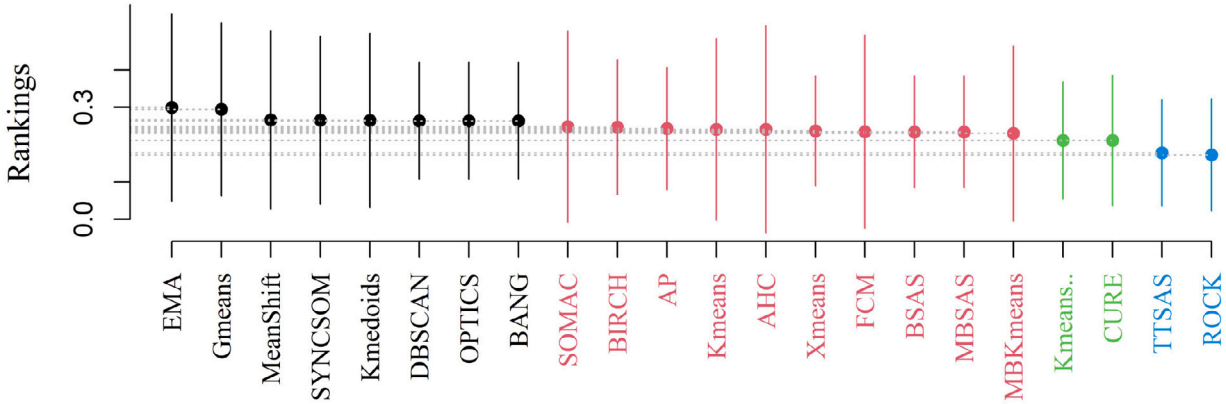

Fig. 26. The Rankings of clustering techniques in terms of PofB@20% on NASA and SOFTLAB datasets. Different colors indicate different ranking levels. The higher ranking value indicates the better performance of each clustering technique on PofB@20%.

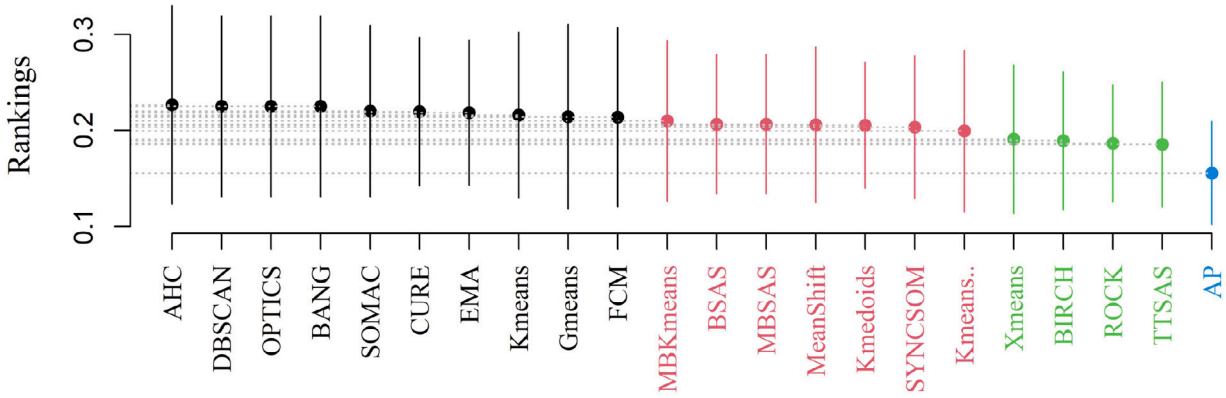

Fig. 27. The Rankings of clustering techniques in terms of PMI@20% on NASA and SOFTLAB datasets. Different colors indicate different ranking levels. The lower ranking value indicates the better performance of each clustering technique on PMI@20%.

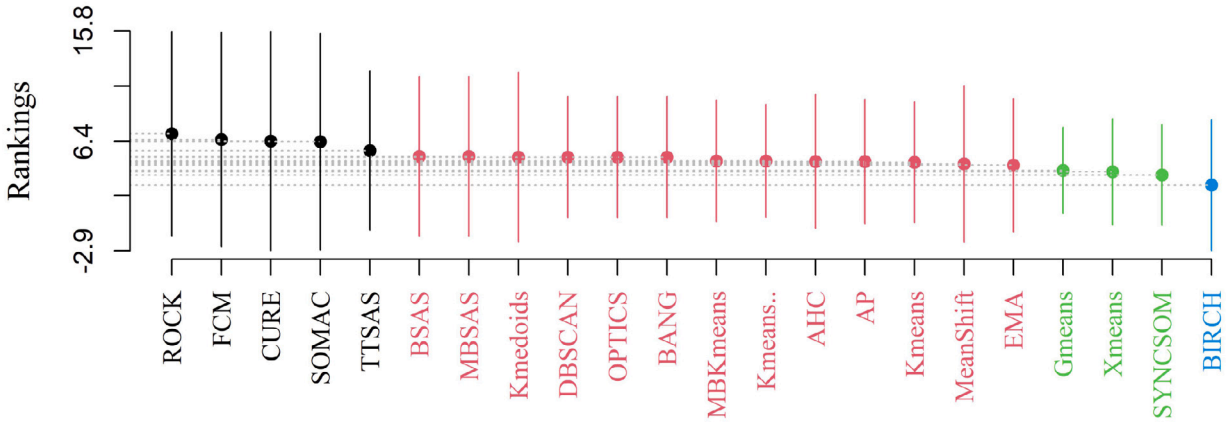

Fig. 28. The Rankings of clustering techniques in terms of IFA on NASA and SOFTLAB datasets. Different colors indicate different ranking levels. The lower ranking value indicates the better performance of each clustering technique on IFA.

unsupervised clustering techniques we have identified, namely K-medoids, Kmeans++, and CURE, to conduct EADP research. Compared to ManualUp, these three unsupervised clustering techniques exhibit superior performance in several key metrics (i.e., PMI@20% and IFA). This implies that applying the three best-performing unsupervised clustering techniques to test software modules for bugs significantly reduce the number of tester attempts to identify the first defective module. It also substantially decreases the time testers spend switching between modules, thus saving testing resources. Furthermore, they do not consistently fall short compared to some supervised EADP methods. They outperform some EADP methods, such as EALR and EATT, in specific metrics. In addition, these clustering technologies provide efficient defect prediction for projects lacking labels in real-world scenarios. By sorting modules based on the LOC feature, testers can prioritize examining smaller modules and optimizing resource allocation.

(2) **The effective unsupervised clustering techniques could potentially find application in other related software engineering domains.** For instance, these techniques might prove valuable in software vulnerability detection. The task of identifying software vulnerabilities

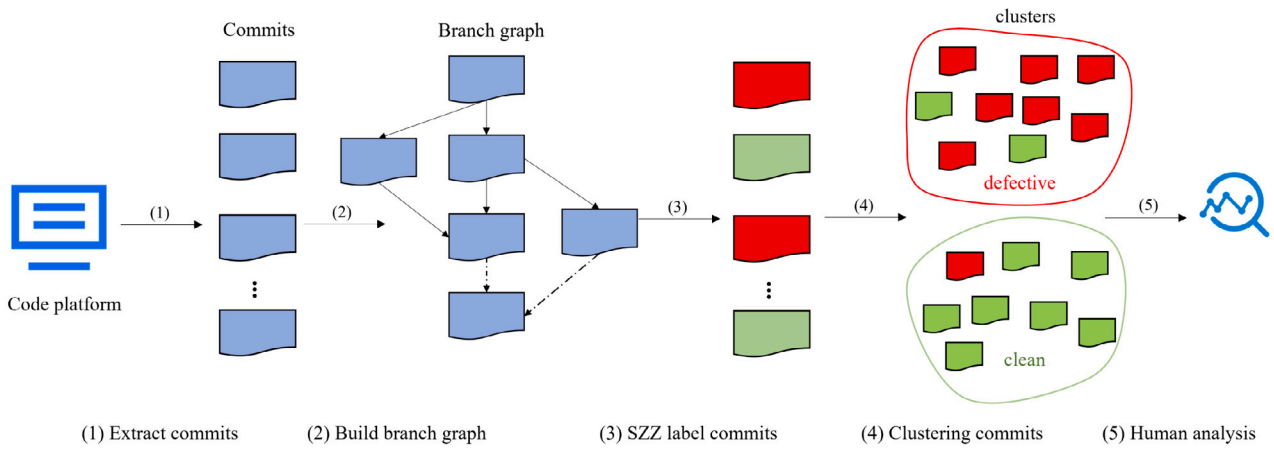

Fig. 29. The process of improved SZZ approach.

is fundamental and critical in fields like information security and software engineering (Croft et al., 2023). Numerous methods employed in vulnerability detection require substantial amounts of well-labeled code fragments as training sets (Lin et al., 2020). However, gathering data for vulnerability detection tasks is often challenging (Xie et al., 2020), making establishing efficient supervised models quite difficult. An alternative approach involves extracting vulnerability-related features from code snippets and then clustering them through the unsupervised techniques recommended in our paper. The clustering process divides code snippets into two clusters, with and without vulnerabilities. Then, according to the LOC characteristics, the code snippet with less LOC is prioritized for detection, so that more vulnerabilities can be detected with less effort. Another potential application of unsupervised clustering algorithms is to enhance the labeling performance of the SZZ approach. Fig. 29 introduces the process of the improved SZZ approach. Specifically, the process begins by extracting commits from the code platform. A branch graph of these commits is established based on the Git version control system. Subsequently, the SZZ technique is employed to label the commits. Following this labeling, clustering techniques are employed to group the commits that have been labeled for the first time. Lastly, commits that were initially labeled as clean but are clustered as defective are subject to manual verification, thereby enhancing the labeling performance of the SZZ approach.

(3) **Not all clustering techniques are effective for unsupervised EADP.** Researchers should utilize the K-medoids method in the field of unsupervised EADP, and refrain from employing the methods of DBSCAN, OPTICS, AP, and BANG. Xu, Li, et al. (2021) investigated the performance of various unsupervised clustering techniques, and they identified several clustering techniques, such as DBSCAN, OPTICS, AP, and ROCK, that performed well on F1@20%. However, these clustering techniques exhibited very high IFA values in our study, which is unacceptable for the testing team. Therefore, their study is not sufficiently persuasive as they lacked exploration of the impact of IFA on unsupervised EADP. Thus, based on their study, we have reexamined 22 clustering techniques. Section 5 displays the results of our experiments, in which the best clustering technique, K-medoids, is able to effectively reduce IFA and PMI@20% compared to ManualUp, EALR, and EATT. K-medoids also demonstrates superior performance on other effort-aware indicators, and are not always inferior to the supervised EADP method. However, according to the results in Section 5, we consider that the methods of DBSCAN, OPTICS, AP, and BANG are unsuitable for unsupervised EADP research, as previous studies have suggested that the IFA must be less than 10.

(4) **Future research should explore better clustering techniques to further enhance the performance of unsupervised EADP.** Section 5 examines the relationship between the classification performance and effort-aware performance of the three best-performing clustering techniques. The experimental results indicate a strong correlation between the classification indicators and the effort-aware indicators, indicating the evident interaction between clustering quality and defect prediction accuracy. This suggests that improving clustering classification capabilities is highly beneficial in unsupervised EADP research. Moreover, based on our experimental results, we find that two of the top-performing unsupervised clustering methods belong to the PBC type. Therefore, we recommend future researchers focus on exploring clustering methods of this type to establish unsupervised EADP models.

8. Conclusion and future works

This paper explores the relative value of clustering techniques for unsupervised EADP. We investigate the impact of 22 clustering techniques for unsupervised EADP on 41 versions of eleven projects from the PROMISE dataset. To provide a comprehensive assessment, we compare these unsupervised clustering techniques with the advanced supervised EADP methods. Firstly, we identify the top three unsupervised clustering techniques. We then subject these methods to a comparison with supervised techniques, including ManualUp, EALR, EATT, CBS+, CBS+(RF), and CBS+(GB). We examine the relationship between the classification performance and effort-aware performance of these three clustering techniques. Finally, we use Precision, Recall, F1, Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, and IFA to evaluate the performance and apply the Scott-Knott ESD test, Wilcoxon signed-rank test, and Kendell correlation coefficient to analyze the experimental results.

- We observe that K-medoids has the best overall performance and can significantly reduce the IFA and PMI@20% values of ManualUp, even better than those of supervised EADP methods EALR and EATT. Therefore, we suggest that researchers use the K-medoids to enhance unsupervised EADP performance further.
- According to the correlation results, we find that better clustering ability can achieve higher effort-aware performance. Therefore, we suggest that future research could focus on exploring better clustering techniques to further improve unsupervised EADP performance.

In the future, our efforts will be directed toward proposing enhanced unsupervised methods to further improve the performance of EADP. Moreover, we will manage to explore approaches for automatically selecting optimal features to rank software modules, even without the use of labels. Meanwhile, we will try to apply these techniques to different software engineering scenarios (e.g., software vulnerability detection) to further explore the possibility of unsupervised techniques.

CRediT authorship contribution statement

Peixin Yang: Data curation, Methodology, Formal analysis, Writing – original draft. **Lin Zhu:** Data curation, Methodology, Writing – original draft. **Yanjiao Zhang:** Resources, Software, Visualization. **Chuanxiang Ma:** Project administration, Software, Validation. **Liming Liu:** Resources, Software, Visualization, Writing – review & editing. **Xiao Yu:** Supervision, Formal analysis, Methodology, Writing – original draft. **Wenhua Hu:** Supervision, Software, Visualization, Writing – review & editing.

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.

Data availability

Data will be made available on request.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (62272356, 61977021) and the Natural Science Foundation of Chongqing, China (cstc2021jcyj-msxmX1115).

References

Ahmadi, N., & Berangi, R. (2008). A basic sequential algorithmic scheme approach for classification of modulation based on neural network. In *2008 international conference on computer and communication engineering* (pp. 565–569). IEEE.

Amasaki, S., Aman, H., & Yokogawa, T. (2022). An evaluation of effort-aware fine-grained just-in-time defect prediction methods. In *2022 48th euromicro conference on software engineering and advanced applications* (pp. 209–216). IEEE.

Angel, L., & Viola, J. (2016). Payload estimation for a robotic system using unsupervised classification. In *2016 XXI symposium on signal processing, images and artificial vision* (pp. 1–5). IEEE.

Askari, S. (2021). Fuzzy C-means clustering algorithm for data with unequal cluster sizes and contaminated with noise and outliers: Review and development. *Expert Systems with Applications*, *165*, Article 113856.

Balaram, A., & Vasundra, S. (2022). Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm. *Automated Software Engineering*, *29*(1), 6.

Bennin, K. E., Keung, J., Monden, A., Kamei, Y., & Ubayashi, N. (2016). Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models. In *2016 IEEE 40th annual computer software and applications conference, vol. 1* (pp. 154–163). IEEE.

Bennin, K. E., Toda, K., Kamei, Y., Keung, J., Monden, A., & Ubayashi, N. (2016). Empirical evaluation of cross-release effort-aware defect prediction models. In *2016 IEEE international conference on software quality, reliability and security* (pp. 214–221). IEEE.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 2.

Bishnu, P. S., & Bhattacherjee, V. (2011). Software fault prediction using quad tree-based k-means clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering*, *24*(6), 1146–1150.

Boetticher, G. (2007). The PROMISE repository of empirical software engineering data. URL: http://promisedata.org/repository.

Çarka, J., Esposito, M., & Falessi, D. (2022). On effort-aware metrics for defect prediction. *Empirical Software Engineering*, *27*(6), 1–38.

Catolino, G., Di Nucci, D., & Ferrucci, F. (2019). Cross-project just-in-time bug prediction for mobile apps: An empirical assessment. In *2019 IEEE/ACM 6th international conference on mobile software engineering and systems* (pp. 99–110). IEEE.

Chen, Y., Huang, J., Mou, L., Jin, P., Xiong, S., & Zhu, X. X. (2023). Deep saliency smoothing hashing for drone image retrieval. *IEEE Transactions on Geoscience and Remote Sensing*, *61*, 1–13.

Chen, Y., Lu, X., & Wang, S. (2020). Deep cross-modal image–voice retrieval in remote sensing. *IEEE Transactions on Geoscience and Remote Sensing*, *58*(10), 7049–7061.

Chen, L.-q., Wang, C., & Song, S.-l. (2022). Software defect prediction based on nested-stacking and heterogeneous feature selection. *Complex & Intelligent Systems*, *8*(4), 3333–3348.

Chen, X., Zhao, Y., Wang, Q., & Yuan, Z. (2018). MULTI: Multi-objective effort-aware just-in-time software defect prediction. *Information and Software Technology*, *93*, 1–13.

Cheng, T., Zhao, K., Sun, S., Mateen, M., & Wen, J. (2022). Effort-aware cross-project just-in-time defect prediction framework for mobile apps. *Frontiers of Computer Science*, *16*(6), 1–15.

Cho, Y., Kwon, J. H., Yi, J., & Ko, I. Y. (2022). Extending developer experience metrics for better effort-aware just-in-time defect prediction. *IEEE Access*, *10*, 128218–128231.

Croft, R., Babar, M. A., & Kholoosi, M. M. (2023). Data quality for software vulnerability datasets. In *2023 IEEE/ACM 45th international conference on software engineering* (pp. 121–133). IEEE.

Deng, D. (2020). DBSCAN clustering algorithm based on density. In *2020 7th international forum on electrical engineering and automation* (pp. 949–953). IEEE.

Ding, C., & He, X. (2002). Cluster merging and splitting in hierarchical clustering algorithms. In *2002 IEEE international conference on data mining, 2002. Proceedings.* (pp. 139–146). IEEE.

Feng, S., Keung, J., Xiao, Y., Zhang, P., Yu, X., & Cao, X. (2024). Improving the undersampling technique by optimizing the termination condition for software defect prediction. *Expert Systems with Applications*, *235*, Article 121084.

Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: Complexity-based OverSampling technique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology*, *129*, Article 106432.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972–976.

Fu, W., & Menzies, T. (2017). Revisiting unsupervised learning for defect prediction. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (pp. 72–83).

Gong, L., Jiang, S., & Jiang, L. (2019). Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access*, *7*, 145725–145737.

Gong, L., Jiang, S., Wang, R., & Jiang, L. (2019). Empirical evaluation of the impact of class overlap on software defect prediction. In *2019 34th IEEE/ACM international conference on automated software engineering* (pp. 698–709). IEEE.

Gong, L., Rajbahadur, G. K., Hassan, A. E., & Jiang, S. (2021). Revisiting the impact of dependency network metrics on software defect prediction. *IEEE Transactions on Software Engineering*, *48*(12), 5030–5049.

Gong, L., Zhang, H., Zhang, J., Wei, M., & Huang, Z. (2022). A comprehensive investigation of the impact of class overlap on software defect prediction. *IEEE Transactions on Software Engineering*, *49*(4), 2440–2458.

Guha, S., Rastogi, R., & Shim, K. (2000). ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, *25*(5), 345–366.

Guha, S., Rastogi, R., & Shim, K. (2001). Cure: an efficient clustering algorithm for large databases. *Information Systems*, *26*(1), 35–58.

Ha, D. A., Chen, T. H., & Yuan, S. M. (2019). Unsupervised methods for software defect prediction. In *Proceedings of the 10th international symposium on information and communication technology* (pp. 49–55).

Herbold, S., Trautsch, A., & Grabowski, J. (2017). Global vs. local models for cross-project defect prediction: A replication study. *Empirical Software Engineering*, *22*, 1866–1902.

Huang, Q., Shihab, E., Xia, X., Lo, D., & Li, S. (2018). Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, *23*, 418–451.

Huang, Q., Xia, X., & Lo, D. (2017). 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* (pp. 159–170). IEEE.

Huang, Q., Xia, X., & Lo, D. (2019). Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering*, *24*(5), 2823–2862.

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Heming, J. (2022). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, http://dx.doi.org/10.1016/j.ins.2022.11.139.

Jin, C. (2021). Cross-project software defect prediction based on domain adaptation learning and optimization. *Expert Systems with Applications*, *171*, Article 114637.

Jing, X., Wu, F., Dong, X., Qi, F., & Xu, B. (2015). Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp. 496–507).

Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th international conference on predictive models in software engineering* (pp. 1–10).

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K. i., Adams, B., & Hassan, A. E. (2010). Revisiting common bug prediction findings using effort-aware models. In *2010 IEEE international conference on software maintenance* (pp. 1–10). IEEE.

Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2012). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, *39*(6), 757–773.

Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software defect prediction analysis using machine learning techniques. *Sustainability*, *15*(6), 5517.

Khatri, Y., & Singh, S. K. (2022). Towards building a pragmatic cross-project defect prediction model combining non-effort based and effort-based performance measures for a balanced evaluation. *Information and Software Technology*, *150*, Article 106980.

Kochhar, P. S., Xia, X., Lo, D., & Li, S. (2016). Practitioners' expectations on automated fault localization. In *Proceedings of the 25th international symposium on software testing and analysis* (pp. 165–176).

Li, D., Liang, M., Xu, B., Yu, X., Zhou, J., & Xiang, J. (2021). A cross-project aging-related bug prediction approach based on joint probability domain adaptation and k-means SMOTE. In *2021 IEEE 21st international conference on software quality, reliability and security companion* (pp. 350–358). IEEE.

Li, F., Lu, W., Keung, J. W., Yu, X., Gong, L., & Li, J. (2023). The impact of feature selection techniques on effort-aware defect prediction: An empirical study. *IET Software*, *17*(2), 168–193.

Li, H., & Wang, J. (2022). Collaborative annealing power k-means++ clustering. *Knowledge-Based Systems*, *255*, Article 109593.

Li, F., Yang, P., Keung, J. W., Hu, W., Luo, H., & Yu, X. (2023). Revisiting 'revisiting supervised methods for effort-aware cross-project defect prediction'. *IET Software*, *17*(4), 472–495.

Li, W., Zhang, W., Jia, X., & Huang, Z. (2020). Effort-aware semi-supervised just-in-time defect prediction. *Information and Software Technology*, *126*, Article 106364.

Li, F., Zou, K., Keung, J. W., Yu, X., Feng, S., & Xiao, Y. (2023). On the relative value of imbalanced learning for code smell detection. *Software - Practice and Experience*, *53*(10), 1902–1927.

Liang, M., Li, D., Xu, B., Zhao, D., Yu, X., & Xiang, J. (2021). Within-project software aging defect prediction based on active learning. In *2021 IEEE international symposium on software reliability engineering workshops* (pp. 1–8). IEEE.

Lin, G., Wen, S., Han, Q. L., Zhang, J., & Xiang, Y. (2020). Software vulnerability detection using deep neural networks: a survey. *Proceedings of the IEEE*, *108*(10), 1825–1848.

Liu, X., Xu, Z., Yang, D., Yan, M., Zhang, W., Zhao, H., Xue, L., & Fan, M. (2022). An unsupervised cross project model for crashing fault residence identification. *IET Software*, *16*(6), 630–646.

Lund, B., & Ma, J. (2021). A review of cluster analysis techniques and their uses in library and information science research: k-means and k-medoids clustering. *Performance Measurement and Metrics*, *22*(3), 161–173.

Ma, X., Keung, J., He, P., Xiao, Y., Yu, X., & Li, Y. (2023). A semi-supervised approach for industrial anomaly detection via self-adaptive clustering. *IEEE Transactions on Industrial Informatics*.

Ma, X., Keung, J., Yang, Z., Yu, X., Li, Y., & Zhang, H. (2022). CASMS: Combining clustering with attention semantic model for identifying security bug reports. *Information and Software Technology*, *147*, Article 106906.

Ma, X., Keung, J. W., Yu, X., Zou, H., Zhang, J., & Li, Y. (2023). AttSum: A deep attention-based summarization model for bug report title generation. *IEEE Transactions on Reliability*.

Majd, A., Vahidi-Asl, M., Khalilian, A., Poorsarvi-Tehrani, P., & Haghighi, H. (2020). SLDeep: Statement-level software defect prediction using deep-learning model on static code features. *Expert Systems with Applications*, *147*, Article 113156.

Mehta, R., Garain, J., & Singh, K. K. (2022). Cohort selection using mini-batch k-means clustering for ear recognition. In *Advances in intelligent computing and communication: Proceedings of ICAC 2021* (pp. 273–279). Springer.

Mende, T., & Koschke, R. (2010). Effort-aware defect prediction models. In *2010 14th European conference on software maintenance and reengineering* (pp. 107–116). IEEE.

Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., & Zimmermann, T. (2012). Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, *39*(6), 822–834.

Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., & Cok, D. (2011). Local vs. global models for effort estimation and defect prediction. In *2011 26th IEEE/ACM international conference on automated software engineering* (pp. 343–351). IEEE.

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., & Bener, A. (2010). Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, *17*, 375–407.

Mughnyanti, M., Efendi, S., & Zarlis, M. (2020). Analysis of determining centroid clustering x-means algorithm with davies-bouldin index evaluation. In *IOP conference series: materials science and engineering, vol. 725, no. 1*. IOP Publishing, Article 012128.

Ni, C., Xia, X., Lo, D., Chen, X., & Gu, Q. (2022). Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. *IEEE Transactions on Software Engineering*, *48*(3), 786–802.

Ni, C., Xia, X., Lo, D., Yang, X., & Hassan, A. E. (2022). Just-in-time defect prediction on JavaScript projects: A replication study. *ACM Transactions on Software Engineering and Methodology*, *31*(4), 1–38.

Novikov, A., & Benderskaya, E. (2014). SYNC-SOM. In *Proceedings of the 3rd international conference on pattern recognition applications and methods* (pp. 305–309).

Öztürk, M. M., Cavusoglu, U., & Zengin, A. (2015). A novel defect prediction method for web pages using k-means++. *Expert Systems with Applications*, *42*(19), 6496–6506.

Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. *Engineering Applications of Artificial Intelligence*, *111*, Article 104773.

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications*, *144*, Article 113085.

Park, M., & Hong, E. (2014). Software fault prediction model using clustering algorithms determining the number of clusters automatically. *International Journal of Software Engineering and Its Applications*, *8*(7), 199–204.

Peng, T., & Nie, Q. (2017). *SOMSC: Self-organization-map for high-dimensional single-cell data of cellular states and their transitions*. Cold Spring Harbor Laboratory, Article 124693, bioRxiv.

Qu, Y., Chi, J., & Yin, H. (2021). Leveraging developer information for efficient effort-aware bug prediction. *Information and Software Technology*, *137*, Article 106605.

Qu, Y., Zheng, Q., Chi, J., Jin, Y., He, A., Cui, D., Zhang, H., & Liu, T. (2019). Using K-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Transactions on Software Engineering*, *47*(2), 348–366.

Ranjbarzadeh, R., & Saadi, S. B. (2020). Automated liver and tumor segmentation based on concave and convex points using fuzzy c-means and mean shift clustering. *Measurement*, *150*, Article 107086.

Rao, J., Yu, X., Zhang, C., Zhou, J., & Xiang, J. (2021). Learning to rank software modules for effort-aware defect prediction. In *2021 IEEE 21st international conference on software quality, reliability and security companion* (pp. 372–380). IEEE.

Sandhu, A. K., & Batth, R. S. (2021). Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm. *Software - Practice and Experience*, *51*(4), 735–747.

Schikuta, E., & Erhart, M. (1998). BANG-clustering: A novel grid-clustering algorithm for huge data sets. In *Advances in pattern recognition: Joint IAPR international workshops SSPR'98 and SPR'98 Sydney, Australia, August 11–13, 1998 Proceedings* (pp. 867–874). Springer.

Shao, Y., Liu, B., Wang, S., & Li, G. (2018). A novel software defect prediction based on atomic class-association rule mining. *Expert Systems with Applications*, *114*, 237–254.

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215.

Shivaji, S., Whitehead, E. J., Akella, R., & Kim, S. (2012). Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, *39*(4), 552–569.

Subudhi, A., Dash, M., & Sabut, S. (2020). Automated segmentation and classification of brain stroke using expectation-maximization and random forest classifier. *Biocybernetics and Biomedical Engineering*, *40*(1), 277–289.

Subudhi, S., & Panigrahi, S. (2022). Application of OPTICS and ensemble learning for database intrusion detection. *Journal of King Saud University-Computer and Information Sciences*, *34*(3), 972–981.

Sudakov, O., & Dmitriev, D. (2022). Comparison of G-means algorithms and kohonen network in solving clustering problems. In *Graphicon-conference on computer graphics and vision, vol. 32* (pp. 1147–1156).

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2018). The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, *45*(7), 683–711.

Theodoridis, S., & Koutroumbas, K. (2006). *Pattern recognition*. Elsevier.

Thirumoorthy, K., & Britto, J. J. J. (2022). A clustering approach for software defect prediction using hybrid social mimic optimization algorithm. *Computing*, *104*(12), 2605–2633.

Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, *14*(5), 540–578.

Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011). 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* (pp. 15–25). ACM.

Xia, X., Lo, D., Pan, S. J., Nagappan, N., & Wang, X. (2016). Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on Software Engineering*, *42*(10), 977–998.

Xiang, C., Yuxiang, S., Shaoqing, M., Zhanqi, C., Xiaolin, J., & Zan, W. (2018). Multi-objective optimization based feature selection method for software defect prediction. *Journal of Frontiers of Computer Science & Technology*, *12*(9), 1420.

Xie, Q., Dai, Z., Hovy, E., Luong, T., & Le, Q. (2020). Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, *33*, 6256–6268.

Xu, Z., Li, L., Yan, M., Liu, J., Luo, X., Grundy, J., Zhang, Y., & Zhang, X. (2021). A comprehensive comparative study of clustering-based unsupervised defect prediction models. *Journal of Systems and Software*, *172*, Article 110862.

Xu, Z., Zhao, K., Zhang, T., Fu, C., Yan, M., Xie, Z., Zhang, X., & Catolino, G. (2021). Effort-aware just-in-time bug prediction for mobile apps via cross-triplet deep feature embedding. *IEEE Transactions on Reliability*, *71*(1), 204–220.

Yan, M., Fang, Y., Lo, D., Xia, X., & Zhang, X. (2017). File-level defect prediction: Unsupervised vs. supervised models. In *2017 ACM/IEEE international symposium on empirical software engineering and measurement* (pp. 344–353). IEEE.

Yang, Z., Keung, J. W., Yu, X., Xiao, Y., Jin, Z., & Zhang, J. (2023). On the significance of category prediction for code-comment synchronization. *ACM Transactions on Software Engineering and Methodology*, *32*(2), 1–41.

Yang, X., Yu, H., Fan, G., & Yang, K. (2020). A differential evolution-based approach for effort-aware just-in-time software defect prediction. In *Proceedings of the 1st ACM SIGSOFT international workshop on representation learning for software engineering and program languages* (pp. 13–16).

Yang, X., Yu, H., Fan, G., & Yang, K. (2021). DEJIT: a differential evolution algorithm for effort-aware just-in-time software defect prediction. *International Journal of Software Engineering and Knowledge Engineering*, *31*(03), 289–310.

Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., & Leung, H. (2016). 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* (pp. 157–168).

Yin, S., Li, H., Liu, D., & Karim, S. (2020). Active contour modal based on density-oriented BIRCH clustering method for medical image segmentation. *Multimedia Tools and Applications*, *79*, 31049–31068.

Yu, X., Bennin, K. E., Liu, J., Keung, J. W., Yin, X., & Xu, Z. (2019). An empirical study of learning to rank techniques for effort-aware defect prediction. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering* (pp. 298–309). IEEE.

Yu, X., Dai, H., Li, L., Gu, X., Keung, J. W., Bennin, K. E., Li, F., & Liu, J. (2023). Finding the best learning to rank algorithms for effort-aware defect prediction. *Information and Software Technology*, *157*, Article 107165.

Yu, X., Keung, J., Xiao, Y., Feng, S., Li, F., & Dai, H. (2022). Predicting the precise number of software defects: Are we there yet? *Information and Software Technology*, *146*, Article 106847.

Yu, X., Liu, J., Keung, J. W., Li, Q., Bennin, K. E., Xu, Z., Wang, J., & Cui, X. (2019). Improving ranking-oriented defect prediction using a cost-sensitive ranking SVM. *IEEE Transactions on Reliability*, *69*(1), 139–153.

Yu, X., Liu, J., Yang, Z., Jia, X., Ling, Q., & Ye, S. (2017). Learning from imbalanced data for predicting the number of software defects. In *2017 IEEE 28th international symposium on software reliability engineering* (pp. 78–89). IEEE.

Yu, X., Rao, J., Hu, W., Keung, J., Zhou, J., & Xiang, J. (2024). Improving effort-aware defect prediction by directly learning to rank software modules. *Information and Software Technology*, *165*, Article 107250.

Yu, X., Wu, M., Jian, Y., Bennin, K. E., Fu, M., & Ma, C. (2018). Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning. *Soft Computing*, *22*, 3461–3472.

Zain, Z. M., Sakri, S., & Ismail, N. H. A. (2023). Application of deep learning in software defect prediction: Systematic literature review and meta-analysis. *Information and Software Technology*, http://dx.doi.org/10.1016/j.infsof.2023.107175.

Zhang, Y., Lo, D., Xia, X., & Sun, J. (2018). Combined classifier for cross-project defect prediction: an extended empirical study. *Frontiers of Computer Science*, *12*, 280–296.

Zhang, F., Zheng, Q., Zou, Y., & Hassan, A. E. (2016). Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proceedings of the 38th international conference on software engineering* (pp. 309–320).

Zhao, K., Xu, Z., Yan, M., Xue, L., Li, W., & Catolino, G. (2022). A compositional model for effort-aware just-in-time defect prediction on android apps. *IET Software*, *16*(3), 259–278.

Zheng, W., Shen, T., Chen, X., & Deng, P. (2022). Interpretability application of the just-in-time software defect prediction model. *Journal of Systems and Software*, *188*, Article 111245.