Automated Software Engineering (2024) 31:19
https://doi.org/10.1007/s10515-024-00424-1

Software defect prediction: future directions and challenges

Zhiqiang Li[1] · Jingwen Niu[2] · Xiao-Yuan Jing[3,4]

Received: 8 September 2023 / Accepted: 5 February 2024 / Published online: 27 February 2024

Abstract
Software defect prediction is one of the most popular research topics in software engineering. The objective of defect prediction is to identify defective instances prior to the occurrence of software defects, thus it aids in more effectively prioritizing software quality assurance efforts. In this article, we delve into various prospective research directions and potential challenges in the field of defect prediction. The aim of this article is to propose a range of defect prediction techniques and methodologies for the future. These ideas are intended to enhance the practicality, explainability, and actionability of the predictions of defect models.

Keywords Software defect prediction · Empirical software engineering · Software analytics · Quality assurance

1 Introduction

Software defect prediction (SDP) is a vibrant research domain in software engineering and plays an important role in ensuring quality assurance (Menzies et al. 2010; Kamei and Shihab 2016; Wan et al. 2020; Tantithamthavorn and Hassan 2018). It is

✉ Zhiqiang Li
lizq@snnu.edu.cn

Jingwen Niu
niujw66@163.com

Xiao-Yuan Jing
jingxy_2000@126.com

1 School of Computer Science, Shaanxi Normal University, Xi'an 710119, China

2 School of Computer and Information Engineering, Xinxiang University, Xinxiang 453003, China

3 School of Computer Science, Wuhan University, Wuhan 430072, China

4 School of Computer, Guangdong University of Petrochemical Technology, Maoming 525000, China

a very crucial and essential activity. By identifying defective instances (e.g., components, files, classes, methods, changes) before testing, SDP has the potential to reduce code inspection costs and improve software quality. This empowers software quality assurance teams to effectively allocate their limited resources for testing and maintenance, leading to enhanced efficiency (Li et al. 2018c).

In the past few decades, analytical modeling of defects has attracted a lot of attention from both the academic and industrial communities (Hall et al. 2012; Hosseini et al. 2019; Chen et al. 2021). Various SDP methods have been introduced across different prediction settings, such as within-project defect prediction (WPDP) (Menzies et al. 2007; Lessmann et al. 2008; Ghotra et al. 2015), cross-project defect prediction (CPDP) (Zimmermann et al. 2009; Li et al. 2021, 2023), heterogeneous defect prediction (HDP) (Jing et al. 2015; Nam and Kim 2015; Li et al. 2018a, 2017, 2018b), and just-in-time defect prediction (Kamei et al. 2013; Zhao et al. 2023). These methods have yielded promising defect prediction results. Therefore, SDP plays a pivotal role in software development organizations worldwide nowadays. Given its significance, we believe that the timing is opportune to write a paper on the future of software engineering focusing on the subject of software defect prediction.

This paper is presented from the perspective of academic researchers and has two primary objectives. Firstly, it provides a brief overview of SDP and highlights several key steps within this domain. Secondly, it proposes a collection of research directions for future defect prediction.

2 Defect prediction process

The common software defect prediction process involves the utilization of various machine learning techniques and methodologies (Shepperd et al. 2014; Li et al. 2018c; Giray et al. 2023), ranging from classic classification algorithms to advanced deep learning architectures, to proactively identify and detect potential defects in software instances. This essential process is vital for enhancing software quality and reliability. It is comprised of several key steps, each playing an important role in predicting and mitigating defects before they manifest in the final software products. Figure 1 illustrates these steps in the software defect prediction process, including data collection, preprocessing, model construction and prediction, and finally the evaluation and interpretation of the defect prediction results. This process is not only integral for ensuring software quality but also plays a crucial role in prioritizing resource allocation and minimizing the impact of defects on users and stakeholders. The following sections provide detailed descriptions for each step.

① *Data collection*. In software defect prediction, data collection involves gathering information about projects, specifically related to code versions, historical changes, and associated attributes. This includes data on past defects, bug reports, source code metrics, process metrics, developer activities, and other relevant software project features (Li et al. 2018c). Alternatively, it might involve directly extracting abstract syntax trees and various types of graphs from source codes for deep learning-based techniques (Giray et al. 2023; Zain et al. 2023). The aim is to create a comprehensive dataset that enables the training and evaluation of machine

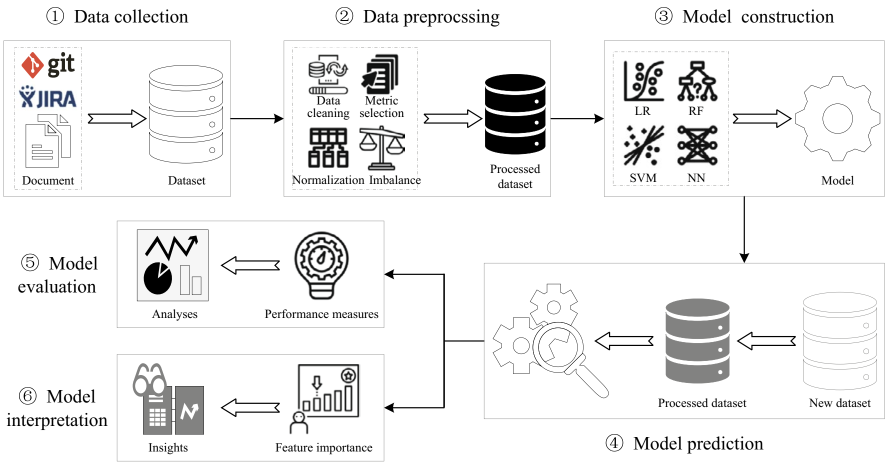

Fig. 1 Overview of the common software defect prediction process

learning models for predicting potential software defects. This data can be acquired from version control systems (e.g., Git, CVS, SVN), bug tracking tools (e.g, JIRA, Bugzilla, GitHub Issues), code repositories, project documentation, and other repositories where relevant information is stored. In short, efficient and accurate data collection is vital for building effective defect prediction models (Kim et al. 2011; Wu et al. 2011; Tantithamthavorn et al. 2015).

② *Data preprocessing*. It is a crucial aspect in the context of defect prediction, involving preparation and manipulation of the raw data collected from software repositories to enhance the performance and accuracy of predictive models in identifying software defects. This process usually includes several key steps. (1) *Data cleaning*: Cleaning and preparing the collected data for defect analytics. It consists of handling missing values, dealing with outliers, addressing data inconsistencies, handing noise, and removing duplicate instances (Shepperd et al. 2013). This step ensures that the dataset used for defect prediction is consistent, accurate, and free from anomalies, setting a solid foundation for building robust and reliable predictive models. (2) *Metric selection*: Identifying and selecting the most relevant features or metrics that are likely to have a correlation with software defects. Meanwhile, eliminating correlated metrics that may not contribute significantly to the model's performance (Jiarpakdee et al. 2021a). This aims to reduce dimensionality and improve the efficiency of the prediction models. (3) *Normalization*: Normalizing or scaling numerical features within a specific range or distribution. It aims to make all the software metrics in a dataset to a similar scale without distorting differences in the ranges of values. The techniques commonly used for data normalization include log transformation (Menzies et al. 2007) and z-score standardization (Li et al. 2019a). These methods adjust the values of metrics to a standardized scale, ensuring consistency across the dataset. (4) *Handling class imbalance*: The defect dataset might have an imbalance between defective and non-defective instances. Employing class imbalance learning techniques, such as the widely used undersampling, oversampling, or synthetic data sampling algorithms, to rebalance the distribution between defective and non-defective instances in the dataset is a common strategy (Tantithamthavorn et al. 2020).

③ *Model construction*. Building a defect prediction model using machine learning techniques on the preprocessed dataset. This step includes selecting an appropriate learning algorithm, training the model on the labeled dataset, and optimizing its parameters to effectively predict the presence of defects in software projects based on various features or metrics. Common algorithms for defect prediction involve logistic regressions, decision trees, random forests, naive Bayes, support vector machines, and neural networks (Ghotra et al. 2015; Tantithamthavorn et al. 2019). The goal is to build a model that generalizes well to new or unseen software instances and accurately predicts whether the given instances contain defects or not.

④ *Model prediction*. Applying the trained model that was constructed during the training phase, to predict the status of new or unseen software instances passed through the model. The model generates predictions or classifications based on the input data, aiming to determine whether these instances are likely to be defective or not. Additionally, it might rank instances based on their predicted density (probability/LOC) to facilitate prioritized inspection in a more cost-effective way (Kamei et al. 2010; Mende and Koschke 2010). This is a crucial step in defect prediction as it helps software developers and quality assurance teams to proactively identify potential defects in software before they cause issues or errors in production, enabling timely actions to improve software quality and reliability.

⑤ *Model evaluation*. Assessing the performance and effectiveness of the constructed defect prediction model based on its outputs. Common non-effort-aware and effort-aware performance measures (Huang et al. 2019; Li et al. 2018c) used for model evaluation include AUC (area under the receiver operating characteristic curve), MCC (Matthew's correlation coefficient), F1-score, PoB@20% (proportion of the found bugs among all bugs in the dataset when inspecting 20% LOC), PMI@20% (proportion of modules inspected when test 20% LOC), and IFA (the number of initial false alarms encountered before software testers detect the first defect). This step is crucial for understanding how well the model performs on new or unseen data, and aids in understanding the model's strengths, weaknesses, and areas for improvement. The goal of model evaluation is to measure the model's effectiveness in making predictions.

⑥ *Model interpretation*. Understanding and explaining how the defect model makes predictions or decisions. Model interpretation aims to provide insights into the model's inner workings, making its output more understandable and transparent to humans. Through thorough analysis and extraction of meaningful insights from predictions, it seeks to identify which software metrics are the most influential in defect prediction. This can assist software practitioners in understanding why the defect prediction model made particular decisions and provides insights into the rationale behind predictions made by analytical models (Dam et al. 2018; Tantithamthavorn and Jiarpakdee (2021)). The interpretative process is indispensable for building trust in defect prediction models, emphasizing that the rationale behind a model's decisions is just as significant as the decisions themselves.

The overall goal of software defect prediction is to help quality assurance teams prioritize their efforts and resource allocation by identifying software instances that are more likely to contain defects. This proactive approach significantly contributes to enhancing software quality, reducing maintenance costs, and improving overall software development practices. Through the early identification of potential defects, defect prediction enables quality assurance teams to allocate their resources more efficiently, paying attention to critical areas and streamlining the quality assurance process. Ultimately, the application of defect prediction techniques brings in a more robust and reliable software development lifecycle, fostering higher-quality deliverables and minimizing the impact of defects on software products.

3 Future directions and challenges

The field of software defect prediction has witnessed numerous achievements over the past decades (Kamei and Shihab 2016; Wan et al. 2020). Nevertheless, it is important to note that many challenges remain and are likely to emerge in the future due to shifts in data, technology, and the ever-growing significance of software systems. To enhance readability, we organize this section into four dimensions, including data, metrics, model construction, and model evaluation and interpretation, which is roughly similar to the defect prediction process introduced in Sect. 2. Fig. 2 illustrates an overview of the future directions and challenges in software defect prediction, with detailed descriptions for each perspective provided below.

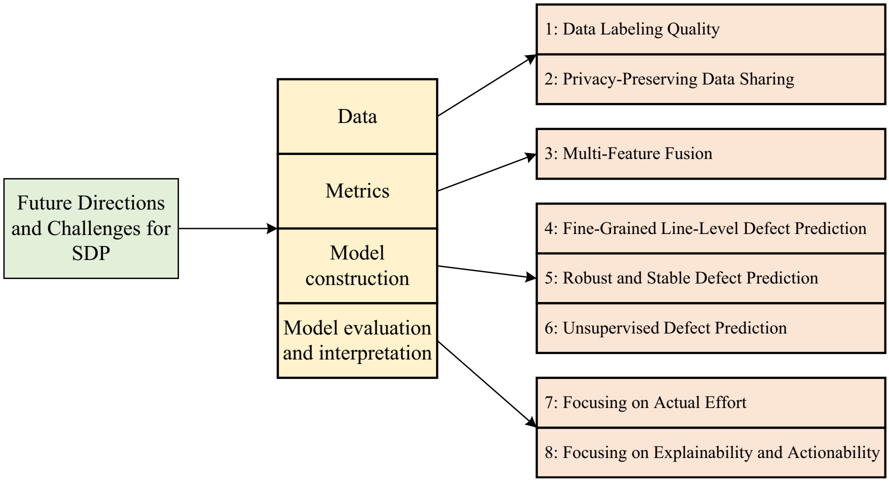

Fig. 2 Overview of future directions and challenges in software defect prediction

3.1 Data

Future direction and challenge 1: Data labeling quality. *Context:* Historical defect information plays a pivotal role in software maintenance such as quality measurement and defect prediction. The SZZ algorithm (Śliwerski et al. 2005) stands as a primary method for identifying bug-inducing commits in software projects, which is widely used in the field of defect prediction. *Issue:* However, current defect collection practices are based on optional bug fix keywords or bug report links in change logs, which can lead to the inclusion of noise into the collected data (Kim et al. 2011; Wu et al. 2011; Tantithamthavorn et al. 2015). Such biased data can significantly impact the performance of defect prediction models. The SZZ algorithm faces issues in achieving high precision due to the presence of noise within bug-fixing commits. For instance, not all addition or deletion lines within a bug-fixing commit are directly linked to bug fixes (Tang et al. 2023). *Direction and challenge:* A potential direction is that substantial efforts are directed towards improving the precision of the SZZ algorithm in the future. Despite the advancements made by the existing studies (Kim et al. 2006; Da Costa et al. 2016; Neto et al. 2018; Tsantalis et al. 2018), it is challenging to incorporate all refactoring and non-essential change patterns into a tool, as this could result in the potential exclusion of relevant lines and the inclusion of irrelevant lines (Tang et al. 2023). In recent years, deep learning techniques have garnered extensive application across various software engineering tasks (Yang et al. 2022; Samoaa et al. 2022), consistently outperforming other state-of-the-art methods. The notable strength of deep learning lies in its ability to autonomously learn highly intricate and expressive features, a capability that traditional methods cannot be done. This advantage allows deep learning models to capture complex patterns and relationships within data more effectively. Therefore, one promising avenue is the utilization of deep learning techniques for the identification of bug-inducing commits. By leveraging the power of deep learning, it becomes possible to automatically capture semantic relationships within commit data that contribute to more effective models, which conventional SZZ algorithms find challenging.

Future direction and challenge 2: Privacy-preserving data sharing. *Context:* In recent years, many researchers have utilized dataset collected from open-source software projects and have demonstrated a willingness to make their data availability and openness to facilitate reproducibility. *Issue:* Numerous commercial and proprietary software projects often lack data availability due to business sensitivity and privacy concerns. This has raised doubts about the feasibility of data sharing for research purposes. Recently, privacy preservation issue has gained attention in the field of defect prediction (Peters et al. 2013, 2015; Li et al. 2019b). *Direction and challenge:* The need for more privacy-preserving data sharing initiatives becomes crucial for further exploration. This can potentially facilitate the availability of more commercial and proprietary data. Benefiting from this, existing methods such as cross-project defect prediction (Zhou et al. 2018) or heterogeneous defect prediction (Chen et al. 2021) could offer practical value, especially for new projects or those lacking enough historical data. However, it is very challenging for many companies or organizations are not willing to share their data due to concerns related to business sensitivity and privacy. To address this challenge, researchers need to establish strong partnerships with industrial collaborators and should actively seek collaboration with them, gaining access to their rich and diverse data repositories. Meanwhile, it is imperative to explore new methods for privacy-preserving data sharing. Lately, Yamamoto et al. (2023) presented a federated logistic regression model for privacy-preserving cross-project defect prediction. Inspired by this, the combination of federated deep learning and other privacy techniques emerges as a promising avenue for preserving and maintaining the security of data in defect prediction. Federated learning (Lo et al. 2021), a decentralized training approach, allows machine learning models to be trained across multiple edge devices without the need for exchanging raw training data. This not only enhances privacy protection but also facilitates collaborative learning across diverse datasets.

3.2 Metrics

Future direction and challenge 3: Multi-feature fusion. *Context:* Defect data typically consists of multiple types of software metrics, e.g., code metrics, process metrics, ownership metrics, etc (Menzies et al. 2007; Moser et al. 2008; Bird et al. 2011). Each type of metric characterizes the relevant attributes of a software product from a certain perspective, which has different physical meanings and distributions. When considering semantic features, various types of source code representations have been proposed (Yang et al. 2022; Samoaa et al. 2022). These contain abstract syntax trees, control flow graphs, call flow graphs, data flow graphs, program dependency graphs, token-based embedding representations and so on. Indeed, these representations have found wide application in various software engineering tasks. *Issue:* For the traditional hand-crafted features, existing defect prediction studies identify software defects by concatenating and merging all the metrics into a single feature vector. For the semantic features, most defect prediction studies use each code representation individually for identifying software defects. However, these methods ignore the diversity and complementary information among different types of metrics or multiple code representations (Zhou et al. 2022; Ni et al. 2022a). *Direction and challenge:* Considering that metric data are extracted from software projects from different perspectives, each type of metric can be recognized as a single data view. So defect data consisting of multiple types of metrics can be divided into multiple different data views. A promising avenue for future research involves fusing multiple types of software metrics to build robust and reliable defect prediction models. Such an approach would enable the collaborative learning of the diversity and complementarity within defect data, which will improve the model performance. However, the process of effectively fusing multiple types of software metrics could indeed pose a significant challenge. Regarding the semantic features, exploring the potential complementarity between tree-based and graph-based code representations in defect prediction could be beneficial for future work, particularly in the context of fusing multiple code representations. However, it also poses challenges in accurately extracting various source code representations, encoding them appropriately, and effectively combining them by using existing off-the-shelf deep learning techniques. This fusion process demands innovative strategies to harness the comprehensive potential of these diverse code representations.

Indeed, some studies (Xu et al. 2020; Wang et al. 2021; Zhou et al. 2022; Ni et al. 2022a) have attempted to fuse traditional manually designed software metrics with semantic features derived from code representations to enhance the accuracy of defect prediction. The results of these studies have indicated that the combination of traditional hand-crafted metrics with semantic features contributes to improved performance in software defect prediction. However, effectively combining diverse software metrics and features presents its set of challenges. The process demands innovative approaches that address issues of feature representation, integration strategies, and the optimal balance between the information learned from various sources. Finding an effective fusion technique that maximizes the strengths of each type of feature while mitigating potential redundancies remains an ongoing challenge in this domain.

3.3 Model construction

Future direction and challenge 4: Fine-grained line-level defect prediction. *Context:* The current defect prediction models typically are at a relatively coarse granularity level, such as the file level. This often makes software practitioners needing to spend significant effort in inspecting many clean lines that are actually non-defective. *Issue:* In practice, practitioners are interested in identifying the specific lines of code that are defective (Wattanakriengkrai et al. 2022). Hence, there is a growing need for defect prediction models to become more fine-grained and capable of pinpointing the truly lines of code that require attention. This finer granularity can significantly enhance the practicality and usefulness of these models in real-world software development and maintenance. However, most of the defect prediction studies did not pay attention to this domain. *Direction and challenge:* It holds promise towards code-line-level defect prediction, as it could help developers to more effectively prioritize their quality assurance efforts. Indeed, there is an increasing recognition of the potential advantages in fine-grained, code-line-level defect prediction (Pornprasit and Tantithamthavorn 2021, 2023; Ni et al. 2022a; Guo et al. 2023). Meanwhile, it is important to note that substantial challenges exist in building accurate code-line-level defect prediction models effectively. In the context of traditional metrics, the primary challenge in constructing conventional defect models at the code line level is the design of manually crafted software features. Extracting such features at the code line level is a challenging endeavor, as it demands precise historical data for each line within the source code files. In the context of semantic features, the collected datasets often remain highly dimensional and sparse. Consequently, building code-line-level defect prediction models using semantic features is likely unfeasible and impractical within a shorter period of time (Wattanakriengkrai et al. 2022). Despite these challenges, the move towards a finer granularity in defect prediction can offer a more targeted and actionable approach, enabling teams to address potential issues at a more localized level within the source code, which aligns with the industry's pursuit of higher precision and efficiency in software quality assurance processes.

Future direction and challenge 5: Robust and stable defect prediction. *Context:* Ensuring robust and stable defect prediction is imperative because software projects are dynamic, and the data used to train models may change over time. Defect models lacking in robustness and stability may yield unreliable predictions, leading to decreased confidence in their effectiveness for identifying software defects. *Issue:* Recent works (Fu et al. 2016; Tantithamthavorn et al. 2019) have highlighted a crucial point regarding defect prediction models in the literature. They argue that most of these models tend to rely on the default parameter settings of classification techniques, which usually have a large impact on the performance of these models and lead to suboptimal results. Thus, the hyperparameters of the defect prediction models should be carefully tuned. However, a critical observation in existing work (Tantithamthavorn et al. 2019) is the limited quantity of hyperparameters explored for the examined classifiers. Most of these classifiers focus on tuning a single parameter, and the parameter space considered is relatively small in scale. *Direction and challenge:* There exists an opportunity for future research to delve into exploring multiple hyperparameters and broader parameter spaces, particularly in the context of deep learning-based defect prediction models. Exploring the interactions among various parameters could provide valuable insights and potentially lead to significant improvements in the performance and effectiveness of defect prediction models. Undoubtedly, it is important to acknowledge that dealing with multiple hyperparameters and expansive search spaces can pose challenges. Striking a balance between comprehensiveness and practicality will be crucial in designing experiments that are both informative and feasible. Nonetheless, the exploration of more extensive and intricate parameter configurations holds promise in achieving robust and stable defect prediction performance.

Future direction and challenge 6: Unsupervised defect prediction. *Context:* Supervised defect prediction trains models on labeled data to predict the occurrence of defects or bugs in software. Instead, unsupervised defect prediction (Li et al. 2020; Xu et al. 2021) builds models on unlabeled data through the application of unsupervised learning techniques for identifying software defects. *Issue:* Although supervised defect prediction methods have the potential to achieve better results in certain performance measures, they do have limitations. The main issue with supervised methods is that they require labeled training data to build models. The process of obtaining labeled data can be time-consuming, labor-intensive, and costly, which makes supervised methods inefficient and resource-intensive, especially in scenarios with limited historical defect data. *Direction and challenge:* Unsupervised defect prediction methods aim to identify more likely defect-prone software instances on unlabeled datasets by exploiting the intrinsic patterns and structures present in the data without the need for any labeled training data. They hold the advantage of not requiring prior knowledge of defect data to label modules. This characteristic makes unsupervised methods particularly beneficial in practice, especially for new software projects or projects with insufficient historical data. For example, Zhang et al. (2016) introduced a spectral clustering method that utilizes connectivity-based unsupervised classifiers for predicting software defects. Their findings demonstrated the superiority of the proposed spectral clustering approach over some supervised classifiers in both within-project and cross-project settings. Hence, unsupervised prediction techniques exhibit considerable potential in field of software defect prediction and represent a promising avenue for future research. However, the challenge remains in devising methods that effectively uncover the intrinsic structures and patterns within the data to achieve more accurate unsupervised defect prediction.

3.4 Model evaluation and interpretation

Future direction and challenge 7: Focusing on actual effort. *Context:* It is of utmost importance to evaluate defect prediction models in a realistic context, e.g. how much effort can reduce for testing and code inspection using these models. By taking into account the resources and efforts required for code inspection or testing, effort-aware defect prediction (Kamei et al. 2010; Mende and Koschke 2010) provides a more accurate assessment of prediction model effectiveness and aligns the evaluations with real-world scenarios. *Issue:* In recent years, there are many effort-aware defect prediction studies (Kamei et al. 2013; Yang et al. 2016; Huang et al. 2019; Ni et al. 2022b) to account for the effort. Typically, these models utilize LOC (lines of code) or churn as a proxy for effort. As pointed out by Shihab et al. (2013), their results show that LOC is not the best measure of effort in effort-aware defect prediction. *Direction and challenge:* To progress in the field, it is crucial for researchers to shift their focus towards the utilization of actual effort data in future effort-aware defect prediction research. Adopting this approach is expected to yield more reliable insights and enhance practical guidelines for software practitioners. One potential strategy involves leveraging effort estimation techniques (Menzies et al. 2013) to calculate actual effort. Effort estimation is the process of predicting the amount of effort required to develop or maintain a software application. Integrating this technique into defect prediction holds significant promise. However, identifying the most effective approach for accurately quantifying real effort and thereby provide robust practical guidelines remains a complex and ongoing challenge. Undoubtedly, such research endeavors can have a substantial impact on enhancing the future applicability of defect prediction techniques in real-world software development practices.

Future direction and challenge 8: Focusing on explainability and actionability. *Context:* In the field of defect prediction, a significant proportion of research efforts have concentrated primarily on improving the predictive accuracy and performance of defect models to more accurately identify potential defects in software systems. The aim is to enhance software quality and minimize the occurrence of defects. *Issue:* The above research efforts have largely neglected or underemphasized comprehensive explanations and justifications (Dam et al. 2018; Tantithamthavorn and Jiarpakdee (2021)). Particularly, current defect prediction fails to explain why models make such a prediction and fails to comply with the privacy laws in terms of the requirement to explain any decision made by a method. A lack of explainability of the defect prediction models leads to a lack of trust in the predictions or recommendations produced by such methods, hindering their widespread adoption in software development practices (Jiarpakdee et al. 2021b). *Direction and challenge:* The future landscape of defect prediction research should strongly emphasize explainability and actionability, particularly when focusing on deep learning-based models. A recent empirical study by Jiarpakdee et al. (2022) delved into evaluating model-agnostic techniques for explaining the predictions generated by defect models. Their findings revealed the utility of generating explanations through model-agnostic techniques for each prediction. Such explanations play a crucial role in aiding developers to comprehend why a file or commit is identified as defective, while simultaneously providing actionable guidance to assist project managers in devising suitable quality improvement plans. In short, the overarching goal is to make defect prediction models more practical, explainable, and actionable in software engineering practices. Nevertheless, it is essential to acknowledge that ensuring the reliability and trustworthiness of these predictions of defect models from the perspective of software practitioners remains an ongoing and challenging endeavor.

4 Conclusion

In this article, we present a several of future directions and potential challenges in software defect prediction. It is important to note that our work does not aim to be exhaustive; rather, it simply serves as a documentation of future directions and potential challenges. Most importantly, we would like to emphasize that we do not seek to claim the generality of our ideas. Instead, the goal of this article is that under specific circumstances, these research directions have the potential to help developers to effectively find software defects and enable managers to better develop software quality improvement plans to prevent defects in the future.

Author contributions Zhiqiang Li: methodology, writing Jingwen Niu: investigation, resources Xiao-Yuan Jing: review, editing. All authors reviewed the manuscript.

Funding National Natural Science Foundation of China (Grant Nos.: 61902228 and 62176069), Natural Science Basic Research Program of Shaanxi Province (Grant No.: 2024JC-YBMS-497), and funded by the China Scholarship Council.

Declarations

Competing interests The authors declare no competing interests.

References

Bird, C., Nagappan, N., Murphy, B., et al.: Don't touch my code!: Examining the effects of ownership on software quality. In: ESEC/FSE'11. ACM, pp. 4–14 (2011)

Chen, H., Jing, X.Y., Li, Z., et al.: An empirical study on heterogeneous defect prediction approaches. IEEE Trans. Softw. Eng. **47**(12), 2803–2822 (2021)

Da Costa, D.A., McIntosh, S., Shang, W., et al.: A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes. IEEE Trans. Softw. Eng. **43**(7), 641–657 (2016)

Dam, H.K., Tran, T., Ghose, A.: Explainable software analytics. In: Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 53–56 (2018)

Fu, W., Menzies, T., Shen, X.: Tuning for software analytics: Is it really necessary? Inf. Softw. Technol. **76**, 135–146 (2016)

Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: ICSE'15. IEEE, pp. 789–800 (2015)

Giray, G., Bennin, K.E., Köksal, Ö., et al.: On the use of deep learning in software defect prediction. J. Syst. Softw. **195**, 111537 (2023)

Guo, Z., Liu, S., Liu, X., et al.: Code-line-level bugginess identification: How far have we come, and how far have we yet to go? ACM Trans. Softw. Eng. Methodol. **32**(4), 1–55 (2023)

Hall, T., Beecham, S., Bowes, D., et al.: A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. **38**(6), 1276–1304 (2012)

Hosseini, S., Turhan, B., Gunarathna, D.: A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. Softw. Eng. **45**(2), 111–147 (2019)

Huang, Q., Xia, X., Lo, D.: Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empir. Softw. Eng. **24**, 2823–2862 (2019)

Jiarpakdee, J., Tantithamthavorn, C., Hassan, A.: The impact of correlated metrics on the interpretation of defect models. IEEE Trans. Softw. Eng. **47**(2), 320–331 (2021)

Jiarpakdee, J., Tantithamthavorn, C.K., Grundy, J.: Practitioners' perceptions of the goals and visual explanations of defect prediction models. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, pp. 432–443 (2021b)

Jiarpakdee, J., Tantithamthavorn, C., Dam, H.K., et al.: An empirical study of model-agnostic techniques for defect prediction models. IEEE Trans. Softw. Eng. **48**(1), 166–185 (2022)

Jing, X., Wu, F., Dong, X., et al.: Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: FSE'15. ACM, pp. 496–507 (2015)

Kamei, Y., Shihab, E.: Defect prediction: Accomplishments and future challenges. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, pp. 33–45 (2016)

Kamei, Y., Matsumoto, S., Monden, A., et al.: Revisiting common bug prediction findings using effort-aware models. In: 2010 IEEE International Conference on Software Maintenance. IEEE, pp. 1–10 (2010)

Kamei, Y., Shihab, E., Adams, B., et al.: A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Softw. Eng. **39**(6), 757–773 (2013)

Kim, S., Zimmermann, T., Pan, K., et al.: Automatic identification of bug-introducing changes. In: 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), IEEE, pp. 81–90 (2006)

Kim, S., Zhang, H., Wu, R., et al.: Dealing with noise in defect prediction. In: ICSE'11, pp. 481–490 (2011)

Lessmann, S., Baesens, B., Mues, C., et al.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. **34**(4), 485–496 (2008)

Li, N., Shepperd, M.J., Yuchen, G.: A systematic review of unsupervised learning techniques for software defect prediction. Inf. Softw. Technol. **122**, 106287 (2020)

Li, Z., Jing, X.Y., Zhu, X., et al.: Heterogeneous defect prediction through multiple kernel learning and ensemble learning. In: ICSME'17. IEEE, pp. 91–102 (2017)

Li, Z., Jing, X.Y., Wu, F., et al.: Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Autom. Softw. Eng. **25**(2), 201–245 (2018)

Li, Z., Jing, X.Y., Zhu, X.: Heterogeneous fault prediction with cost sensitive domain adaptation. Softw. Test. Verif. Reliab. **28**(2), 1–22 (2018)

Li, Z., Jing, X.Y., Zhu, X.: Progress on approaches to software defect prediction. IET Softw. **12**(3), 161–175 (2018)

Li, Z., Jing, X.Y., Zhu, X., et al.: Heterogeneous defect prediction with two-stage ensemble learning. Autom. Softw. Eng. **26**(3), 599–651 (2019)

Li, Z., Jing, X.Y., Zhu, X., et al.: On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Trans. Softw. Eng. **45**(4), 391–411 (2019)

Li, Z., Niu, J., Jing, X.Y., et al.: Cross-project defect prediction via landmark selection-based kernelized discriminant subspace alignment. IEEE Trans. Reliab. **70**(3), 996–1013 (2021)

Li, Z., Zhang, H., Jing, X.Y., et al.: Dssdpp: data selection and sampling based domain programming predictor for cross-project defect prediction. IEEE Trans. Softw. Eng. **49**(4), 1941–1963 (2023)
Lo, S.K., Lu, Q., Wang, C., et al.: A systematic literature review on federated machine learning: from a software engineering perspective. ACM Comput. Surv. **54**(5), 1–39 (2021)
Mende, T., Koschke, R.: Effort-aware defect prediction models. In: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, pp. 107–116 (2010)
Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. **33**(1), 2–13 (2007)
Menzies, T., Milton, Z., Turhan, B., et al.: Defect prediction from static code features: current results, limitations, new approaches. Autom. Softw. Eng. **17**(4), 375–407 (2010)
Menzies, T., Butcher, A., Cok, D., et al.: Local versus global lessons for defect prediction and effort estimation. IEEE Trans. Softw. Eng. **39**(6), 822–834 (2013)
Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: ICSE'08. IEEE, pp. 181–190 (2008)
Nam, J., Kim, S.: Heterogeneous defect prediction. In: FSE'15. ACM, pp. 508–519 (2015)
Neto, E.C., Da Costa, D.A., Kulesza, U.: The impact of refactoring changes on the SZZ algorithm: an empirical study. In: 2018 IEEE 25th International Conference on Software Analysis, pp. 380–390. Evolution and Reengineering (SANER), IEEE (2018)
Ni, C., Wang, W., Yang, K., et al.: The best of both worlds: integrating semantic features with expert features for defect prediction and localization. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 672–683 (2022a)
Ni, C., Xia, X., Lo, D., et al.: Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans. Softw. Eng. **48**(3), 786–802 (2022)
Peters, F., Menzies, T., Gong, L., et al.: Balancing privacy and utility in cross-company defect prediction. IEEE Trans. Softw. Eng. **39**(8), 1054–1068 (2013)
Peters, F., Menzies, T., Layman, L.: Lace2: Better privacy-preserving data sharing for cross project defect prediction. In: ICSE'15, pp. 801–811 (2015)
Pornprasit, C., Tantithamthavorn, C.K.: Jitline: a simpler, better, faster, finer-grained just-in-time defect prediction. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), IEEE, pp 369–379 (2021)
Pornprasit, C., Tantithamthavorn, C.K.: Deeplinedp: towards a deep learning approach for line-level defect prediction. IEEE Trans. Softw. Eng. **49**(1), 84–98 (2023)
Samoaa, H.P., Bayram, F., Salza, P., et al.: A systematic mapping study of source code representation for deep learning in software engineering. IET Softw. **16**(4), 351–385 (2022)
Shepperd, M., Song, Q., Sun, Z., et al.: Data quality: some comments on the NASA software defect datasets. IEEE Trans. Softw. Eng. **39**(9), 1208–1215 (2013)
Shepperd, M., Bowes, D., Hall, T.: Researcher bias: the use of machine learning in software defect prediction. IEEE Trans. Softw. Eng. **40**(6), 603–616 (2014)
Shihab, E., Kamei, Y., Adams, B., et al.: Is lines of code a good measure of effort in effort-aware models? Inf. Softw. Technol. **55**(11), 1981–1993 (2013)
Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? ACM SIGSOFT Softw. Engi. Notes **30**(4), 1–5 (2005)
Tang, L., Bao, L., Xia, X., et al.: Neural SZZ algorithm. In: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 1024–1035 (2023)
Tantithamthavorn, C., Hassan, A.E.: An experience report on defect modelling in practice: Pitfalls and challenges. In: Proceedings of the 40th International conference on software engineering: software engineering in practice, pp. 286–295 (2018)
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., et al.: The impact of mislabelling on the performance and interpretation of defect prediction models. In: ICSE'15. IEEE, pp. 812–823 (2015)
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., et al.: The impact of automated parameter optimization on defect prediction models. IEEE Trans. Softw. Eng. **45**(7), 683–711 (2019)
Tantithamthavorn, C., Hassan, A.E., Matsumoto, K.: The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Trans. Softw. Eng. **46**(11), 1200–1219 (2020)
Tantithamthavorn, C.K., Jiarpakdee, J.: Explainable ai for software engineering. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp. 1–2 (2021)

Tsantalis, N., Mansouri, M., Eshkevari, L.M., et al.: Accurate and efficient refactoring detection in commit history. In: Proceedings of the 40th International Conference on Software Engineering, pp. 483–494 (2018)
Wan, Z., Xia, X., Hassan, A.E., et al.: Perceptions, expectations, and challenges in defect prediction. IEEE Trans. Softw. Eng. **46**(11), 1241–1266 (2020)
Wang, H., Zhuang, W., Zhang, X.: Software defect prediction based on gated hierarchical LSTMS. IEEE Trans. Reliab. **70**(2), 711–727 (2021)
Wattanakriengkrai, S., Thongtanunam, P., Tantithamthavorn, C., et al.: Predicting defective lines using a model-agnostic technique. IEEE Trans. Softw. Eng. **48**(5), 1480–1496 (2022)
Wu, R., Zhang, H., Kim, S., et al.: Relink: recovering links between bugs and changes. In: FSE/ESEC'11, pp 15–25 (2011)
Xu, J., Wang, F., Ai, J.: Defect prediction with semantics and context features of codes based on graph representation learning. IEEE Trans. Reliab. **70**(2), 613–625 (2020)
Xu, Z., Li, L., Yan, M., et al.: A comprehensive comparative study of clustering-based unsupervised defect prediction models. J. Syst. Softw. **172**(3), 110862 (2021)
Yamamoto, H., Wang, D., Rajbahadur, G.K., et al.: Towards privacy preserving cross project defect prediction with federated learning. In: 2023 IEEE International Conference on Software Analysis, pp. 485–496. Evolution and Reengineering (SANER), IEEE (2023)
Yang, Y., Zhou, Y., Liu, J., et al.: Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: FSE'16, pp 157—168 (2016)
Yang, Y., Xia, X., Lo, D., et al.: A survey on deep learning for software engineering. ACM Comput. Surv. **54**(10s), 1–73 (2022)
Zain, Z.M., Sakri, S., Ismail, N.H.A.: Application of deep learning in software defect prediction: systematic literature review and meta-analysis. Inf. Softw. Technol. **158**, 107175 (2023)
Zhang, F., Zheng, Q., Zou, Y., et al.: Cross-project defect prediction using a connectivity-based unsupervised classifier. In: ICSE'16, pp 309–320 (2016)
Zhao, Y., Damevski, K., Chen, H.: A systematic survey of just-in-time software defect prediction. ACM Comput. Surv. **55**(10), 1–35 (2023)
Zhou, C., He, P., Zeng, C., et al.: Software defect prediction with semantic and structural information of codes based on graph neural networks. Inf. Softw. Technol. **152**, 107057 (2022)
Zhou, Y., Yang, Y., Lu, H., et al.: How far we have progressed in the journey? An examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. **27**(1), 1–51 (2018)
Zimmermann, T., Nagappan, N., Gall, H., et al.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: FSE/ESEC'09. ACM, pp 91–100 (2009)