IET Software

Review Article

IET Journals
The Institution of Engineering and Technology

Progress on approaches to software defect prediction

ISSN 1751-8806
Received on 22nd June 2017
Revised 17th January 2018
Accepted on 10th February 2018
E-First on 12th March 2018
doi: 10.1049/iet-sen.2017.0148
www.ietdl.org

Zhiqiang Li[1], Xiao-Yuan Jing[1,2] ✉, Xiaoke Zhu[1,3]

[1]*School of Computer, Wuhan University, Wuhan 430072, People's Republic of China*
[2]*School of Automation, Nanjing University of Posts and Telecommunications, Nanjing 210023, People's Republic of China*
[3]*School of Computer and Information Engineering, Henan University, Kaifeng 475001, People's Republic of China*
✉ *E-mail: jingxy_2000@126.com*

Abstract: Software defect prediction is one of the most popular research topics in software engineering. It aims to predict defect-prone software modules before defects are discovered, therefore it can be used to better prioritise software quality assurance effort. In recent years, especially for recent 3 years, many new defect prediction studies have been proposed. The goal of this study is to comprehensively review, analyse and discuss the state-of-the-art of defect prediction. The authors survey almost 70 representative defect prediction papers in recent years (January 2014–April 2017), most of which are published in the prominent software engineering journals and top conferences. The selected defect prediction papers are summarised to four aspects: machine learning-based prediction algorithms, manipulating the data, effort-aware prediction and empirical studies. The research community is still facing a number of challenges for building methods and many research opportunities exist. The identified challenges can give some practical guidelines for both software engineering researchers and practitioners in future software defect prediction.

1 Introduction

Software defect prediction is one of the most active research areas in software engineering and plays an important role in software quality assurance [1–5]. The growing complexity and dependency of the software have increased the difficulty in delivering a high quality, low cost and maintainable software, as well as the chance of creating software defects. Software defect usually produces incorrect, or unexpected results and behaviours in unintended ways [6].

Defect prediction is a very crucial and essential activity. Using defect predictors can reduce the costs and improve the software quality by recognising defect-prone modules (instances) prior to testing, such that software engineers can effectively optimise the allocation of limited resources for testing and maintenance. Software defect prediction [7, 8] can be done by classifying a software instance (e.g. method, class, file, change or package level) as defective or non-defective. The defect prediction model can be built by using various software metrics and historical defect data collected from previous release of the same projects [9, 10] or even other projects [11–13]. Such a model will be trained to predict software instances as defective or not. With the prediction model, software engineers can effectively allocate the available testing resources on the defective instances for improving software quality in the early phases of development life cycle. For example, if only 25% of testing resources are available, then software engineers can focus these testing resources on fixing the instances that are more prone to defects. Therefore, a high quality, low cost and maintainable software can be deployed in the given time, resources and budget. That is why, today software defect prediction is a popular research topic in the software engineering field [14].

In the past few decades, more and more research works pay attention to the software defect prediction and a lot of papers have been published. There have already been several excellent systematic review works for the software defect prediction [1, 3–5]. Catal and Diri [3] reviewed 74 defect prediction papers in 11 journals and several conference proceedings by focusing on software metrics, datasets and methods to build defect prediction models. Later on, according to the publication year, Catal [4] investigated 90 defect prediction papers published between year 1990 and year 2009. They mainly survey machine learning and statistical analysis-based methods for defect prediction. Hall *et al.* [1] performed a systematic literature review to investigate how the context of the models, the independent variables used, and the modelling techniques applied influence the performance of defect models. Their analysis is based on 208 defect prediction studies published from January 2000 to December 2010. Recently, Malhotra [5] conducted a systematic review of studies in the literature that use the machine learning techniques in the existing research for software defect prediction. They identified 64 primary studies and seven categories of the machine learning techniques from January 1991 to October 2013. As a summary, the defect prediction published papers are complex and disparate, thus no up-to-date comprehensive picture of the current state of defect prediction exists. These review works are not able to cover the latest progress of the software defect prediction research.

To fill up these gaps of existing systematic review works, this paper tries to provide a comprehensive and systematic review for pioneering works of software defect prediction in recent three years (i.e. from January 2014 to April 2017). A summary of the approaches can be used by researchers as foundation for future investigations of defect prediction. Fig. 1 shows the number of published paper with search keywords ‘‘defect prediction’ OR ‘fault prediction’ OR ‘bug prediction’ AND ‘software’’ on four important computer science libraries: ACM, IEEE, Elsevier and Springer, respectively, from 2014 to 2016. It can be seen that the published papers are gradually increasing and this indicates that the research trend on the topic of defect prediction is growing. Naturally, it is impossible to complete review of all papers as shown in Fig. 1. Similar to prior works [1, 3–5], we need to make some criteria. In this paper, we excluded papers which do not closely related to the research topic on defect prediction, repeated studies and papers not belong to current research hotspot by manual check and screening. Thus, we carefully choose almost 70 representative defect prediction studies in recent years, and most of which are published in the prominent software engineering journals and top conferences. Through this paper, we provide an in-depth study of the major techniques, research hotspots and trends of software defect prediction.

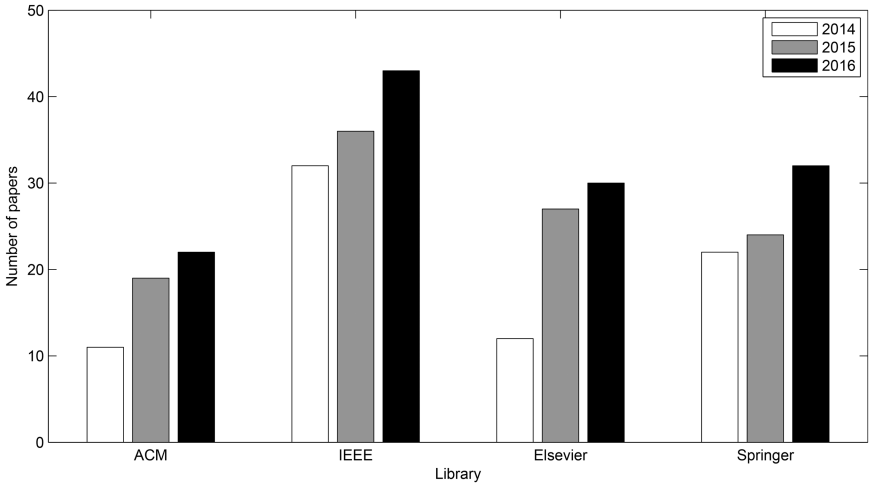

Fig. 1 *Published papers related to software defect prediction on ACM, IEEE, Elsevier and Springer libraries*

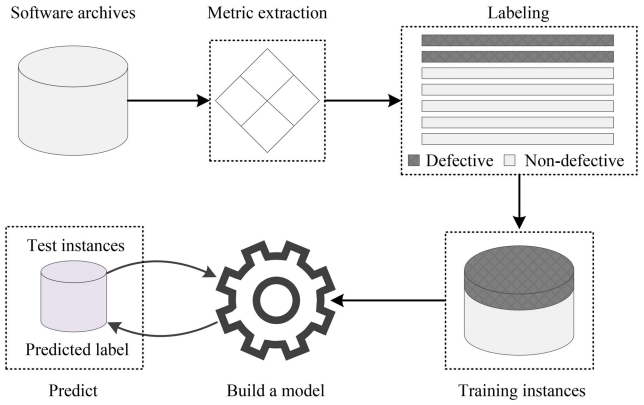

Fig. 2 *Software defect prediction process*

The long history of defect prediction has led to proposal of many theories, approaches, and models. Previous literature reviews [1, 3–5] covered defect prediction studies from 1990 to 2013. Within this article, we perform a detailed analysis of the defect prediction studies between 2014 and 2017. The above literature reviews can provide a comprehensive picture for existing defect prediction studies and cover a long-term progress in this field. Based on these reviews, we can identify and analyse the research trends of defect prediction. Previous literature reviews have performed systematic review in the broader area of defect prediction including modelling techniques, methods, metrics, datasets and performance evaluation measures. In this paper, we aim to summarise and analyse the defect prediction studies that are focusing on the research hotspots and emerging topics, e.g. machine learning-based prediction algorithms, manipulating the data, effort-aware prediction and empirical studies. In this respect, we hope that this review will provide a reference point for conducting future research and yield many high-quality studies in the defect prediction field.

This paper is organised as follows: In Section 2, we briefly introduce the defect prediction process, common software metrics, public datasets and widely used evaluation measures. Section 3 describes the categories of defect prediction algorithms and discusses some new representative techniques. Section 4 reviews various defect prediction studies that focus on manipulating the data. Sections 5 and 6 separately survey the effort-aware context-based defect prediction methods and empirical studies. We outline some future directions and challenges in Section 7 and present conclusions in Section 8.

2 Background

In this section, we firstly briefly introduce the overview of software defect prediction process. Secondly, we briefly review the common software metrics used in defect prediction studies. Thirdly, we present some publicly available benchmark datasets for defect prediction. Finally, we describe the widely used performance evaluation measures in the defect prediction literature.

2.1 Defect prediction process

Most of existing software defect prediction studies have employed machine learning techniques [15–20]. The overview of software defect prediction process based on machine learning classification models is shown in Fig. 2.

To build a defect prediction model, the first step is to create data instances from software archives such as version control systems (e.g. SVN, CVS, GIT), issue tracking systems (e.g. Bugzilla, Jira) and so on. The version control systems contain the source codes and some commit messages, while the issue tracking systems include some defect information. According to prediction granularity, each instance can represent a method, a class, a source code file, a package or a code change. The instance usually contains a number of defect prediction metrics (features), which are extracted from the software archives. The metric values represent the complexity of software and its development process. An instance can be labelled as defective or non-defective according to whether the instance contains defects or not. Then, based on the obtained metrics and labels, the defect prediction models can be built by using a set of training instances. Finally, the prediction model can classify whether a new test instance is defective or not.

2.2 Defect prediction metrics

Software defect prediction metrics play the most important role to build a prediction model that aims to improve software quality by predicting as many software defects as possible. Most defect prediction metrics can be divided into code and process metrics [21]. Code metrics represent how the source code is complex while process metrics represent how the development process is complex [7, 21, 22]. A systematic literature review about the defect prediction metrics can be found in the work [23].

Code metrics are directly collected existing source code and mainly measure properties of the source code such as size and complexity. Its ground assumption is that the source code with higher complexity can be more defect prone. In the past few decades, various code metrics (CMs) have been presented to measure code complexity. For example, lines of code (LOC) metric is one of the commonly used and representative size metrics. According to the number of operators and operands, Halstead [24] presented a few complexity metrics. In order to measure the complexity of software code structure, McCabe [25] proposed several cyclomatic metrics. Afterwards, with object-oriented (OO) programming is getting popular, CMs for OO languages have been presented to improve development process. Chidamber and Kemerer (CK) metrics [26] are some of the most representative metrics for OO programmes. Beyond CK metrics, researchers have presented other OO metrics based on volume and quantity of

source code [27]. These OO metrics simply count the number of variables, methods, classes and so on.

Process metrics are extracted from historical information archived in different software repositories such as version control and issue tracking systems. These metrics reflect the changes over time and quantify many aspects of software development process such as changes of source code, the number of code changes, developer information and so on. In recent years, a number of representative process metrics have been proposed. These metrics can be categorised into the following five groups: (i) code change-based metrics such as relative code change churn [28], change [29], change entropy [30], CM churn, code entropy [31]; (ii) developer information-based metrics such as the number of engineers [32], developer-module networks [33], developer network and social network [34], ownership and authorship [35, 36], developer focus and ownership [37], defect patterns of developer [38], micro interaction metrics [39]; (iii) dependency analysis-based metrics such as dependency graph [40], socio-technical networks [41], change coupling [42], citation influence topic [43], change genealogy [44]; (iv) project team organisation-based metrics such as organisational structure, organisational volatility [45–47]; (v) other process metrics such as popularity [48], anti-pattern [49].

Table 1 shows the common CMs and process metrics for software defect prediction. Both CMs and process metrics are able to build defect prediction models. However, there are different debates on whether CMs are good defect predictors or process metrics are better than CMs. Menzies *et al.* [7] confirmed that CMs are still useful to build defect prediction models based on NASA dataset. Zhang [50] found that it can be used to predict defects well by simply considering the LOC metric. Moser *et al.* [29] conducted a comparative analysis of the predictive power of code and process metrics for defect prediction. They observed that process metrics are more efficient defect predictors than CMs for the Eclipse dataset. Recently, Rahman *et al.* [21] performed an empirical study to compare code and process metrics. They concluded that CMs is less useful than process metrics because of stagnation of CMs.

2.3 Public datasets

Software defect dataset is one of the most important problems for conducting the defect prediction. In the early stages, some academic researchers and companies employed the non-public datasets such as proprietary projects to develop defect prediction models. However, it is not possible to compare the results of such methods to each other, due to their datasets cannot be obtained. Since machine learning researchers had similar problems in 1990s, they created a repository called University of California Irvine (UCI) Machine Learning Repository. Inspired by the success of UCI repository, researchers create PROMISE repository of empirical software engineering data, which has collected several publicly available datasets since 2005. In addition, there are some researchers [31, 44, 51–55] who spontaneously publish their extracted datasets for further empirical study on software defect prediction. In this section, we briefly introduce existing publicly available and commonly used benchmark datasets. Table 2 shows the detailed description of the publicly available datasets.

NASA benchmark dataset consists of 13 software projects [7, 56]. The number of instances ranges from 127 to 17,001, while the number of metrics ranges from 20 to 40. Each project in NASA represents a NASA software system or sub-system, which contains the corresponding defect-marking data and various static CMs. The repository records the number of defects for each instance by using a bug tracking system. Static CMs of NASA datasets include size, readability, complexity attributes and so on, which are closely related to software quality.

Turkish software dataset (SOFTLAB) consists of five projects, which are embedded controller software for white goods [12]. The projects of SOFTLAB are obtained from PROMISE repository and they have 29 metrics. The number of instances ranges from 36 to 121.

Jureczko and Madeyski [57] collected some open source, proprietary and academic software projects, which are part of the PROMISE repository. The collected data consists of 92 versions of 38 different software development projects, including 48 versions of 15 open-source projects, 27 versions of six proprietary projects and 17 academic projects. Each project has 20 metrics in total which contains McCabe's cyclomatic metrics, CK metrics and other OO metrics.

Datasets in ReLink were collected by Wu *et al.* [51] to improve the defect prediction performance by increasing the quality of the defect data. The defect information in ReLink has been manually verified and corrected. ReLink consists of three projects and each one has 26 complexity metrics.

AEEEM was used to benchmark different defect prediction models and collected by D'Ambros *et al.* [31]. Each AEEEM dataset consists of 61 metrics: 17 source CMs, five previous-defect metrics, five entropy-of-change metrics, 17 entropy-of-source-CMs, and 17 churn-of-source CMs [16].

Just-in-time (JIT) dataset was used to study on predicting defect-inducing changes at the change level and collected by Kamei *et al.* [52]. It consists of six open source projects and each project has 14 change metrics including diffusion, size, purpose, history and experience five dimensions.

ECLIPSE1 bug data set was collected by Zimmermann *et al.* [53]. It contains defect data of three Eclipse releases (i.e. 2.0, 2.1 and 3.0), which is extracted on both file and package levels. There are 31 static CMs on the file level and 40 metrics on the package level. The resulting data set lists the number of pre-release and post-release defects for every file and package in these three Eclipse releases.

ECLIPSE2 bug data set was used to study on dealing with the noise in defect prediction and collected by Kim *et al.* [54]. This dataset contains two projects (SWT and Debug) from Eclipse 3.4. The defect data is collected by mining the Eclipse Bugzilla and CVS repositories. There are 17 metrics in total, which contains four different type of metrics, including complexity, OO, change and developer.

The NetGene dataset was used to study the predictive effectiveness of change genealogies in defect prediction and collected by Herzig *et al.* [44]. This dataset consists of four open source projects. Each project has a total of 456 metrics, including complexity metrics, network metrics and change genealogy metrics related to the history of a file.

Table 1 Summary of software metrics

Type	Categories	Representatives
CMs	size	LOC
	complexity	Halstead, McCabe
	OO	CK, other OO metrics
process metrics	code change	relative code change churn, change, change entropy, CM churn, code entropy
	developer information	the number of engineers, developer-module networks, develop network and social network, ownership and authorship, developer focus and ownership, developer defect patterns, micro interaction metrics
	dependency analysis	dependency graph, socio-technical networks, change coupling, citation influence topic, change genealogy
	project team organisation	organisational structure, organisational volatility
	others	popularity, anti-pattern

Table 2 Summary of the publicly available datasets for software defect prediction

Dataset	Description	Number of projects	Metric used	Number of metrics	Granularity	Website
NASA	NASA Metrics Data Program	13	CMs	ranges 20 to 40	function	http://openscience.us/repo/
SOFTLAB	Software Research Laboratory from Turkey	5	CMs	29	function	http://openscience.us/repo/
PROMISE	open source, proprietary and academic projects	38	CMs	20	class	http://openscience.us/repo/
ReLink	recovering links between bugs and changes	3	CMs	26	file	http://www.cse.ust.hk/scc/ReLink.htm
AEEEM	open source projects for evaluating defect prediction models	5	CMs, process metrics	61	class	http://bug.inf.usi.ch/
JIT	open source projects for JIT prediction	6	process metrics	14	change	http://research.cs.queensu.ca/kamei/jittse/jit.zip
ECLIPSE1	eclipse projects for predicting defects	3	CMs	31, 40	file, package	https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/
ECLIPSE2	eclipse projects for handling noise in defect prediction	2	CMs, process metrics	17	file	https://code.google.com/archive/p/hunkim/wikis/HandlingNoise.wiki
NetGen	open source projects for predicting defects using change genealogies	4	CMs, process metrics	465	file	https://hg.st.cs.uni-saarland.de/projects/cg_data_sets/repository
AEV	industry projects for predicting defects	3	CMs, process metrics	29	file	http://www.ist.tugraz.at/_attach/Publish/AltingerHarald/MSR_2015_dataset_automotive.zip

Table 3 Confusion matrix

	Predicted defective	Predicted non-defective
actual defective	TP	FN
actual non-defective	FP	TN

Table 4 Commonly used performance evaluation measures

Measure	Defined as
Pd/recall/TP rate	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
Pf/FP rate	$\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$
precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
F-measure	$\frac{2\times \mathrm{Pd}\times \mathrm{precision}}{\mathrm{Pd}+\mathrm{precision}}=\frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$
G-measure	$\frac{2\times \mathrm{Pd}\times(1-\mathrm{Pf})}{\mathrm{Pd}+(1-\mathrm{Pf})}$
balance	$1-\frac{\sqrt{(0-\mathrm{Pf})^2+(1-\mathrm{Pd})^2}}{\sqrt{2}}$
accuracy	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}}$
G-mean	$\sqrt{\mathrm{Pd}\times(1-\mathrm{Pf})}$
MCC	$\frac{\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})\times(\mathrm{TP}+\mathrm{FN})\times(\mathrm{TN}+\mathrm{FP})\times(\mathrm{TN}+\mathrm{FN})}}$
AUC	the area under the ROC curve
P_{opt}	the area under effort-based cumulative lift charts, which compares a predicted model with an optimal model
AUCEC	the area under cost-effectiveness curve

AEV data set was collected by Altinger *et al.* [55]. It is a novel industry dataset obtained from three different automotive embedded software projects developed by Audi Electronics Venture GmbH. Each project has a total of 29 software metrics.

2.4 Evaluation measures

For defect prediction performance, various evaluation measures have been widely used in [58–62]. The measurement of prediction performance is usually based on the analysis of data in a confusion matrix. This matrix reports how a prediction model classified the different defect categories compared to their actual classification. Table 3 shows the confusion matrix with four defect prediction results. Here, true positive (TP), false negative (FN), false positive (FP) and true negative (TN) are the number of defective instances that are predicted as defective, the number of defective instances that are predicted as non-defective, the number of non-defective instances that are predicted as defective, and the number of non-defective instances that are predicted as non-defective, respectively.

With the confusion matrix, we can define the following performance evaluation measures, which are commonly used in the defect prediction studies. Table 4 shows the mostly used performance evaluation measures for defect prediction.

Pd, probability of detection or *recall* or *true positive rate* is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$. It denotes the ratio of the number of defective instances that are correctly classified as defective to the total number of defective instances. This measure is very important for defect prediction, because prediction models intend to find out defective instances as many as possible.

Pf, probability of false alarm or *false positive rate* is defined as $\mathrm{FP}/(\mathrm{FP}+\mathrm{TN})$. It denotes the ratio of the number of non-defective instances that are wrongly classified as defective to the total number of non-defective instances.

Precision is defined as $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$. It denotes the ratio of the number of defective instances that are correctly classified as defective to the number of instances that are classified as defective. As observed by Menzies *et al.* [63], *precision* is highly unstable performance indicators when data sets contain a low percentage of defects.

F-measure is a comprehensive measure for harmonic mean of Pd and *Precision*. It is defined as $2\times\mathrm{Pd}\times\mathrm{Precision}/(\mathrm{Pd}+\mathrm{Precision})$.

Similar to *F-measure*, the *G-measure* [64] is harmonic mean of Pd and $1-\mathrm{Pf}$, which is defined as $2\times\mathrm{Pd}\times(1-\mathrm{Pf})/(\mathrm{Pd}+(1-\mathrm{Pf}))$. The $1-\mathrm{Pf}$ represents *specificity* [59].

Balance [7] is defined as $1 - \left(\sqrt{(0 - \text{Pf})^2 + (1 - \text{Pd})^2}/\sqrt{2}\right)$. It combines Pf and Pd and calculates by the Euclidean distance from the ROC (receiver operating characteristic curve) sweet spot Pf = 0 and Pd = 1. Hence, better and higher balances fall closer to the desired sweet spot of Pf = 0 and Pd = 1.

Accuracy is defined as (TP + TN)/(TP + FP + FN + TN), which denotes the percentage of correctly predicted instances.

Geometric mean (*G-mean*) [59, 65] is utilised for the overall evaluation of predictors in the imbalanced context. It computes the geometric mean of Pd and 1 − Pf, which is defined as $\sqrt{\text{Pd} \times (1 - \text{Pf})}$.

Matthews correlation coefficient (MCC) [66, 67] measures the correlation between the observed and predicted binary classification with values in [−1, 1]. A value of 1 denotes a perfect prediction, 0 no better than random prediction and −1 represents total disagreement between observation and prediction.

AUC is the area under the ROC curve. This curve is plotted in a two-dimensional space with Pf as *x*-coordinate and Pd as *y*-coordinate. The AUC is known as a useful measure for comparing different models and widely used because AUC is unaffected by class imbalance as well as being independent from the prediction threshold. Other measures such as Pd and *precision* can vary according to prediction threshold values. However, AUC considers prediction performance in all possible threshold values. The higher AUC represents better prediction performance and the AUC of 0.5 means the performance of a random predictor [68].

P_{opt} [52, 60] and AUCEC [68] are two effort-aware measures. Similar to the AUC metric, $P_{\text{opt}} = 1 - \Delta_{\text{opt}}$, where Δ_{opt} is defined as the area between the predicted model and the optimal model. AUCEC is defined as the area under cost-effectiveness curve of the predicted model. These measures consider the LOC to be inspected or tested by quality assurance teams or developers. In fact, both of them can be equivalent. This curve is plotted in a two-dimensional space with *recall* as *x*-coordinate and LOC as *y*-coordinate. The idea of cost-effectiveness for defect prediction models has practical significance in practice. Cost-effectiveness means how many defects can be found among top *n*% LOC inspected or tested. That is, if a certain prediction model can find more defects with less inspecting and testing effort comparing to other models, we could say the cost-effectiveness of the model is higher.

Besides these commonly used evaluation measures, there are some of the less common evaluation measures such as overall error-rate, *H*-measure [69], *J*-coefficient [59] and Brier score [70].

3 Categories of prediction algorithms

Machine learning techniques are the most popular methods for defect prediction [1, 3, 5]. Since new machine learning techniques have been developed, various algorithms such as dictionary learning [71], collaborative representation learning [72], multiple kernel ensemble learning [73], deep learning [74, 75] and transfer learning [76, 77] have been applied to build better defect prediction models. From the view of machine learning, we roughly divide related defect prediction literature into the following three categories: supervised, semi-supervised and unsupervised methods. Supervised learning methods are referred to the use of all labelled training data in a project to build defect prediction models. Semi-supervised learning methods construct defect prediction models by employing only a small number of labelled training data and large number of unlabelled data in a project. Unsupervised learning methods do not require labelled training data, they are directly utilise the unlabelled data in a project to learn defect prediction models.

3.1 Supervised methods

By utilising a recently developed dictionary learning technique, Jing *et al.* [71] designed a cost-sensitive discriminative dictionary learning (CDDL) approach to predict software defect. CDDL exploits class information of historical data to improve the discriminant power and assigns different misclassification costs by increasing punishment on type II misclassification (i.e. defective class is predicted as non-defective) to address the class imbalance problem.

At the same time, Jing *et al.* [72] presented a collaborative representation classification (CRC)-based defect prediction (CSDP) method. CSDP makes use of the recently proposed CRC technique, which main idea is that an instance can be collaboratively represented by a linear combination of all other instances.

Wang *et al.* [73] presented a software defect prediction method, named multiple kernel ensemble learning (MKEL). MKEL can better represent the defect data in a high-dimensional feature space through multiple kernel learning and reduce the bias caused by the non-defective class through assembling a series of weak classifiers. Besides, MKEL specially designs a new sample weight vector updating strategy to relieve the class imbalance problem.

To bridge the gap between programmes' semantics and defect prediction features, Wang *et al.* [74] automatically learned semantic representation of programmes from source code by utilising deep learning techniques. They firstly extracted programmes' abstract syntax trees to obtain the token vectors. Based on the token vectors, they leveraged deep belief network to automatically learn semantic features.

With the utilisation of popular deep learning techniques, Yang *et al.* [75] presented a *Deeper* method for JIT defect prediction. *Deeper* first employs the deep belief network to extract a set of expressive metrics from an initial set of change metrics and then trains a classifier with the extracted metrics to predict defects.

Chen *et al.* [76] designed a novel double transfer boosting (DTB) algorithm for cross-company prediction. DTB firstly adopts data gravitation to reconstruct the distribution of cross-company (CC) data that close to within-company (WC) data. Then, DTB utilises the transfer boosting learning technique to remove negative instances in CC data by using a limited number of WC data.

Xia *et al.* [77] developed a cross-project defect prediction method, named hybrid model reconstruction approach (HYDRA), which contains two phases: genetic algorithm (GA) phase and ensemble learning phase. At the end of these two phases, HYDRA creates a massive composition of classifiers that can be applied to predict defect instances in the target project.

From the view of optimisation, Canfora *et al.* [78] treated the defect prediction problem as a multi-objective optimisation problem. They presented to use GA for training logistic regression and decision tree models, called multi-objective defect predictor (MODEP). MODEP allows software engineers to select predictors reaching a specific trade-off, i.e. the cost of code inspection and the number of defect-prone instances or the number of defects that MODEP can predict.

Ryu *et al.* [79] presented a transfer cost-sensitive boosting (TCSBoost) method to deal with the class imbalance problem for cross-project defect prediction. Based on the distributional characteristics, TCSBoost assigns different misclassification costs for the correct/incorrect classification instances in each iteration of the boosting algorithm.

Yang *et al.* [80] leveraged decision tree and ensemble learning techniques to design a two-layer ensemble learning (TLEL) method for JIT defect prediction. TLEL firstly builds a random forest model by combining decision tree and bagging. With the utilisation of random under-sampling algorithm, TLEL then trains multiple different random forest models and assembles them once more with stacking ensemble.

3.2 Semi-supervised methods

By utilising and combining semi-supervised learning and ensemble learning, Wang *et al.* [81] presented a non-negative sparse-based SemiBoost (NSSB) method for software defect prediction. On one hand, NSSB makes better use of a large number of unlabelled instances and a small number of labelled instances through semi-supervised learning. On the other hand, NSSB assembles a number of weak classifiers to reduce the bias caused by the non-defective class through ensemble learning.

With the utilisation of graph-based semi-supervised learning and sparse representation learning techniques, Zhang *et al.* [82]

proposed a non-negative sparse graph-based label propagation (NSGLP) method for defect prediction. NSGLP firstly resamples the labelled non-defective instances to generate a balanced training dataset. Then, NSGLP constructs the weights by using non-negative sparse graph algorithm to better learn the data relationship. Finally, NSGLP iteratively predicts the unlabelled instances through a label propagation algorithm.

3.3 Unsupervised methods

It is a challenging problem to enable defect prediction for new projects or projects without sufficient historical data. To address this limitation, Nam and Kim [83] presented clustering instances and labelling instances in clusters (CLA) and clustering instances, labelling instances, metric selection and instance selection (CLAMI) two unsupervised methods. The key idea of these two methods is to label an unlabelled dataset by using the magnitude of metric values. Hence, CLA and CLAMI have the advantages of automated manner and no manual effort required.

From the perspective of data clustering, Zhang *et al.* [84] presented to leverage spectral clustering (SC) to tackle the heterogeneity between source project and target project in cross-project defect prediction. SC is a connectivity-based unsupervised clustering method, which does not require any training data. Thus, SC does not suffer from the problem of heterogeneity. This is an advantage of unsupervised clustering methods.

Summary: By comparing and analysing the above defect prediction methods based on machine learning techniques as shown in Table 5, we can obtain following observations: (i) Defect prediction methods usually directly employ or modify existing well-known machine learning algorithms for different prediction contexts. (ii) Most defect prediction methods leverage supervised learning techniques to build classification models. Generally, supervised defect prediction methods are able to improve the prediction performance, especially the classification accuracy. (iii) Most defect prediction methods are based on software CMs, possible because they are easy to collect as compared with process metrics.

4 Manipulating the data

In this section, we review the related defect prediction studies from the perspective of manipulating the data. Specifically, we compare and analyse these studies in the following three aspects: attributes for prediction, data adoption and dataset quality.

4.1 Attributes for prediction

4.1.1 New software metrics: Software metrics, such as source CMs, change churns, and the number of previous defects, have been actively studied to enable defect prediction and facilitate software quality assurance. Recently, there are several methods to design new software metrics for defect prediction [39, 85–87].

Considering developer behaviour, Lee *et al.* [39] leveraged developer interaction information to propose micro-interaction metrics (MIMs). They separately used MIMs, source CMs and change history metrics to build defect prediction models and compared their prediction performance. They found that MIMs significantly improve overall defect prediction accuracy by combining with existing software metrics and perform well in a cost-effective context. The findings can help software engineers to better identify their own inefficient behaviours during software development.

To measure the quality of source code, Okutan and Yildiz [85] introduced a new metric called lack of coding quality (LOCQ). They applied Bayesian networks to examine the probabilistic influential relationships between software metrics and defect-proneness. They found that response for class, lines of code, and LOCQ are the most effective metrics whereas coupling between objects, weighted method per class, and lack of cohesion of methods are less effective metrics for predicting defects.

To explore the predictive ability of mutation-aware defect prediction, Bowes *et al.* [86] defined 40 mutation metrics. They separately built defect prediction models to compare the effectiveness of these mutation-aware metrics and 39 source CMs (mutation-unaware) as well as combining them together. They found that mutation-aware metrics can significantly improve defect prediction performance.

To investigate the effect of concerns on software quality, Chen *et al.* [87] approximated software concerns as topics by using a statistical topic modelling technique. They proposed a set of topic-based metrics including number of topics, number of defect-prone topics, topic membership and defect-prone topic membership. Results show that topic-based metrics provide additional explanatory power over existing structural and historical metrics, which can help better explain software defects.

4.1.2 Feature selection: Feature selection (metric selection) has become the focus of machine learning and data mining, which has also been used in software defect prediction [88, 89]. The aim of feature selection is to select the features which are more relevant to the target class from high-dimensional features and remove the features which are redundant and uncorrelated. After feature selection, the classification performance of prediction models will be improvement.

To check the positive effects of combining feature selection and ensemble learning on the performance of defect prediction, Laradji *et al.* [90] presented an average probability ensemble (APE) method. APE can alleviate the effects caused by metric redundancy and data imbalance on the defect prediction performance. They found that it is very necessary to carefully select relevant and informative features for accurate defect prediction.

Liu *et al.* [91] designed a new feature selection framework, named feature clustering and feature ranking (FECAR). FECAR firstly partitions original features into k clusters, and then selects

Table 5 Comparison of the defect prediction methods using machine learning techniques

Study	Category	Technique	Dataset	Year
supervised	Jing *et al.* [71]	dictionary learning, cost-sensitive learning	NASA	2014
	Jing *et al.* [72]	collaborative representation	NASA	2014
	Wang *et al.* [73]	multiple kernel ensemble learning, boosting	NASA	2016
	Wang *et al.* [74]	deep learning	PROMISE	2016
	Yang *et al.* [75]	deep learning	JIT	2015
	Chen *et al.* [76]	transfer learning, boosting	PROMISE	2015
	Xia *et al.* [77]	transfer learning, GA, AdaBoost	PROMISE	2016
	Canfora *et al.* [78]	multiobjective optimisation, GA	PROMISE	2015
	Ryu *et al.* [79]	boosting, cost-sensitive learning, transfer learning	PROMISE	2017
	Yang *et al.* [80]	decision tree, bagging, random forest	JIT	2017
semi-supervised	Wang *et al.* [81]	SemiBoost, graph learning, sparse representation	NASA	2016
	Zhang *et al.* [82]	sparse representation, sparse graph, label propagation	NASA	2017
unsupervised	Nam and Kim [83]	cluster, feature selection	ReLink	2015
	Zhang *et al.* [84]	spectral clustering	NASA, AEEEM, PROMISE	2016

relevant features from each cluster. Results show that it is effective of FECAR to select features for defect prediction.

To study feature selection methods with a certain noise tolerance, Liu *et al.* [92] presented a feature selection method feature clustering with selection strategies (FECS). FECS contains two phases: feature clustering phase and feature selection phase. They found that FECS is effective on both noise free and noisy datasets.

Xu *et al.* [93] presented a feature selection framework, named maximal information coefficient with hierarchical agglomerative clustering (MICHAC). MICHAC firstly ranks candidate features to filter out irrelevant ones by using maximal information coefficient. Then MICHAC groups features with hierarchical agglomerative clustering and removes redundant features by selecting one feature from each resulted group.

Summary: Table 6 shows the comparison of defect prediction studies that focus on designing new software metrics and related feature selection techniques. Software metrics are very important to build defect prediction models. In the past decades, many effective metrics have been proposed, especially the CMs and process metrics. To facilitate software quality assurance, new defect prediction metrics are being increasingly designed. However, there may be some redundant and irrelevant metrics which are harmful to the prediction performance of learned predictors. To address this problem, feature selection techniques can be used to remove those redundant and irrelevant metrics with the aim of improving the performance.

4.2 Data adoption

4.2.1 Local versus global models: Menzies *et al.* [94] were the first to present local and global models for defect prediction and effort estimation. Local models are referred to clustering the whole dataset into smaller subsets with similar data properties, and building prediction models by training them on these subsets. On the contrary, global model is trained on the whole dataset.

To investigate the performance of local, global and multivariate adaptive regression splines (MARS) three methods, Bettenburg *et al.* [95] separately built defect prediction models for them. Specifically, they constructed local models by splitting the data into smaller homogeneous subsets and learned several individual statistical models, one for each subset. They then built a global model on the whole dataset. Besides, they treated MARS as a global model with local considerations. In terms of modelling the fit and predictive performance, they found that local model can significantly outperform the global model. For practical applications, they observed that MARS produce general trends and can be considered a hybrid between global and local models.

Herbold *et al.* [96] performed a large case of study on local models in the context of cross-project defect prediction. They evaluated and compared the performance of local model, global model and a transfer learning method for cross-project defect prediction. Results show that local models have only a small difference in terms of the prediction performance as compared with the global model and transfer learning method for cross-project defect prediction.

Mezouar *et al.* [97] compared local and global defect prediction models in the effort-aware context. They found that there is at least one local model outperforms the global model, but there always exists another local model performs worse than the global model. They further observed that the worse local model is trained on that the subset of data has a low percentage of defect. Based on these findings, they recommended that files with smaller size should be special attention in future effort-aware defect prediction studies and it is beneficial to combine the advantages of global models by taking local considerations into account in practice.

4.2.2 Heterogeneous data: Heterogeneous defect prediction refers to predicting defect-proneness of software instances in a target project by using heterogeneous metric data collected from other projects. It provides a new perspective to defect prediction and has received much research interest. Recently, several heterogeneous defect prediction models [66, 98–100] are proposed to predict defects across projects with heterogeneous metrics sets (i.e. source and target projects have different metric sets).

Since the metrics except common metrics might have favourable discriminant ability, Jing *et al.* [66] proposed a heterogeneous defect prediction method, namely CCA+ (canonical correlation analysis), which utilises unified metric representation (UMR) and CCA-based transfer learning technique. Specifically, the UMR consists of three types of metrics, including the common metrics of the source and target projects, source-project-specific metrics, and target-project-specific metrics. By learning a pair of projective transformations under which the correlation of the source and target project is maximised, CCA+ can make the data distribution of target project be similar to that of source project.

At the same time, Nam and Kim [98] presented another solution for heterogeneous defect prediction. They firstly employed the metric selection technique to remove redundant and irrelevant metrics for source project. Then, they matched up (e.g. Kolmogorov–Smirnov test-based matching, KSAnalyser) the metrics of source and target projects based on metric similarity such as distribution or correlation. After these processes, they finally arrived at a matched source and target metric sets. With the obtained metric sets, they built heterogeneous defect prediction model to predict labels of the instances in a target.

Table 6 Comparison of the defect prediction studies that handling the metrics

Category	Study	Topic	Technique	Dataset	Year
new metrics	Lee *et al.* [39]	predictive ability evaluation of the micro interaction metrics	correlation-based feature subset selection, random forest	Eclipse system	2016
	Okutan and Yildiz [85]	evaluation of the lack of coding quality metric	Bayesian networks	PROMISE	2014
	Bowes *et al.* [86]	predictive ability evaluation of the mutation metrics	naive Bayes, logistic regression, random forest, J48	3 real word systems	2016
	Chen *et al.* [87]	explanatory power evaluation of topic-based metrics	latent Dirichlet allocation, logistic regression, Spearman correlation analysis	4 real word systems	2017
feature selection	Laradji *et al.* [90]	validation of combining feature selection and ensemble learning	APE, weighted support vector machines	NASA	2015
	Liu *et al.* [91]	methods to build and evaluate feature selection techniques	FF-correlation, FC-relevance	NASA, Eclipse	2014
	Liu *et al.* [92]	evaluation of cluster-based feature selection method with a certain noise tolerance	*k*-medoids clustering, heuristic selection strategy	NASA, Eclipse	2015
	Xu *et al.* [93]	methods to build and evaluate feature selection techniques	hierarchical agglomerative clustering, maximal information coefficient	NASA, AEEEM	2016

He *et al.* [99] presented a CPDP with imbalanced feature sets to address the problem of heterogeneous metric sets in cross-project defect prediction. They used distribution characteristics vectors [13] of each instance as new metrics to enable defect prediction.

To deal with the class imbalance problem under heterogeneous cross-project defect prediction setting, Cheng *et al.* [100] presented a cost-sensitive correlation transfer support vector machine (CCT-SVM) method based on CCA+ [66]. Specifically, to alleviate the influence of imbalanced data, they employed different misclassification costs for defective and non-defective classes by incorporating the cost factors into the support vector machine (SVM) model.

Summary: Table 7 shows the comparison of defect prediction studies by focusing on the data adoption. Local versus global models investigate how to find training data with similar distribution for test data. It usually can achieve better prediction performance for training and test data with similar distribution [12, 16]. In practice, human labelling for a large number of unlabelled modules is costly and time consuming, and may not be perfectly accurate. Hence, it is difficult to collect defect information to label a dataset for training a prediction model. Heterogeneous defect prediction has good potential to use all heterogeneous data of software projects for defect prediction on new projects or projects with limited historical defect data [66, 98]. However, the main challenge of heterogeneous defect prediction is to overcome the data distribution differences between source and target projects. Thus, it needs to reshape the distribution of source data to be similar to the target data. In short, it is very useful and interesting to develop high-quality heterogeneous defect prediction methods for future research.

4.3 Dataset quality

4.3.1 Handling class imbalance: Software defect datasets are often highly imbalanced [65, 101, 102]. That is, the number of defective instances (minority) is usually much fewer than the number of non-defective (majority) ones. It is challenging for most conventional classification algorithms to work with data that has an imbalanced class distribution. The imbalanced distribution could cause misclassification of the instances in the minority class, and this is an important factor accounting for the unsatisfactory prediction performance [1, 61]. Recently, more and more researchers have paid attention to the class imbalance problem of defect prediction [69, 103–105].

To deal with the within-project and cross-project class imbalance problem simultaneously, Jing *et al.* [103] developed a unified defect prediction framework. They first utilised the improved subclass discriminant analysis (ISDA) to achieve balanced subclasses for addressing within-project imbalanced data classification. Then, they made the distributions of source and target data similar through semi-supervised transfer component analysis (SSTCA), and combined SSTCA with ISDA to propose the SSTCA + ISDA approach for cross-project class imbalance learning.

To handle the class imbalance problem in the context of online change defect prediction, Tan *et al.* [104] proposed to leverage resampling and updatable classification techniques. They adopted several resampling methods including simple duplicate, synthetic minority over-sampling technique (SMOTE), spread subsample, resampling with/without replacements to increase the percentage of defective instances in the training set for addressing the imbalanced data challenge.

Ryu *et al.* [69] investigated the feasibility of using the class imbalance learning for cross-project defect prediction. They designed a boosting-based model named value cognitive boosting with support vector machine (VCB-SVM). It sets similarity weights according to distributional characteristics, and combines the weights with the asymmetric misclassification cost designed by the boosting algorithm for appropriate resampling.

By utilising class overlap reduction and ensemble imbalance learning techniques, Chen *et al.* [105] presented a new software defect prediction method, called neighbour cleaning learning (NCL) + ensemble random under-sampling (ERUS). On one hand, they eliminated the overlapping non-defective instances through NCL algorithm. On the other hand, they resampled non-defective instances to construct balanced training subsets through ERUS algorithm. Finally, they assembled a series of classifiers to build the ensemble prediction model through adaptive boosting (AdaBoost).

Wu *et al.* [106] presented a cost-sensitive local collaborative representation (CLCR) method for defect prediction. CLCR explores the neighbourhood information among software instances to enhance the prediction ability of the model and incorporates different cost factors for defective and non-defective classes to handle the class imbalance problem.

Liu *et al.* [107] presented a new two-stage cost-sensitive learning method for defect prediction, which employs cost information both in feature selection and classification two stages. In the first stage, they designed three cost-sensitive feature selection algorithms by utilising different misclassification costs. In the second stage, they employed cost-sensitive back propagation neural network classification algorithm with threshold-moving strategy.

Rodriguez *et al.* [108] handled the imbalanced data by comparing and evaluating different types of algorithms for software defect prediction. They separately used sampling, cost-sensitive, ensembles and their hybrid forms to build prediction models. Results show that these techniques can enhance the correct classification of the defective class and the preprocessing of data affects the improvement of prediction model.

Malhotra and Khanna [109] compared three data sampling techniques (resample with replacement, spread subsample,

Table 7 Comparison of the defect prediction studies focusing on the data adoption

Category	Study	Topic	Technique	Dataset	Year
local versus global models	Bettenburg *et al.* [95]	fit ability comparison of local and global models	clustering, MARS s, linear regression	six open source projects	2015
	Herbold *et al.* [96]	predictive ability comparison of local and global models in a cross-project context	clustering, support vector machine	NASA, AEEEM, PROMISE	2017
	Mezouar *et al.* [97]	predictive ability comparison of local and global models	k-medoids clustering, spectral clustering	AEEEM, PROMISE	2016
heterogeneous defect prediction models	Jing *et al.* [66]	a method to solve heterogeneous metric problem	canonical correlation analysis, transfer learning	NASA, SOFTLAB, ReLink, AEEEM	2015
	Nam and Kim [98]	a method to solve heterogeneous metric problem	metric selection, metric matching, transfer learning	NASA, SOFTLAB, ReLink, AEEEM, PROMISE	2015
	He *et al.* [99]	feasibility using different metrics	statistics, logistic regression	ReLink, AEEEM, PROMISE	2014
	Cheng *et al.* [100]	an improved method to solve heterogeneous metric problem	canonical correlation analysis, transfer learning, support vector machine	NASA, SOFTLAB, ReLink, AEEEM	2016

SMOTE) and cost-sensitive MetaCost learners with seven different cost ratios for change prediction. They employed six machine learning classifiers to conduct change prediction under ten-fold cross validation and inter-release validation two settings. They found that resample with replacement sampling technique is better than other techniques.

4.3.2 Handling noises: Defect information plays an important role in software maintenance such as measuring quality and predicting defects. Since current defect collection practices are based on optional bug fix keywords or bug report links in change logs, the collected defect data from change logs and bug reports could include noises [51, 54, 110, 111]. These biased defect data will affect defect prediction performance.

To examine the characteristic of mislabelling, the impact of realistic mislabelling on the prediction performance and the interpretation of defect models, Tantithamthavorn *et al.* [112] conducted an in-depth study of 3931 manually curated issue reports from two large open-source systems for defect prediction. They found that issue report mislabelling is not random and it has rarely impact on the precision of defect prediction models as well as it does not heavily influence the most influential metrics.

Herzig *et al.* [113] investigated the impact of tangled code changes on defect prediction models. They found that up to 20% of all bug fixing changes consisted of multiple tangled, which would severely influence bug counting and bug prediction models. Due to tangled changes, there is at least 16.6% of all source files are incorrectly associated with bug reports for predicting bug-prone files. Experimental results on tangled bug datasets show that untangling tangled code changes can result in more accurate models. They recommended that future case studies can explore better change organisation to reduce the impact of tangled changes.

4.3.3 Privacy-preserving data sharing: Due to privacy concerns, most companies are not willing to share their data. For these reasons, many researchers doubt the practicality of data sharing for the purposes of research. Recently, privacy preservation issue has been investigated in some software engineering applications, e.g. software defect prediction [64, 114], software effort estimation [115] and so on.

To deal with the privacy preservation problem in multi-party scenario for cross-project defect prediction, Peters *et al.* [116] presented LACE2 (large-scale assurance confidentiality environment). LACE2 combines CLIFF, LeaF and MORPH algorithms together. With LACE2, data owners can incrementally add data to a private cache and contribute 'interesting' data that are not similar to the current content of the private cache. Results show that LACE2 can produce higher privacy and provide better defect prediction performance.

Summary: To better clarify and compare the differences among these defect prediction studies, we provide a brief summary in Table 8. The above-mentioned defect prediction studies mainly pay attention to the dataset quality including class imbalance, data noise and privacy-preserving data sharing problems. These studies explore various model techniques and provide some profound implications. As mentioned above in Section 3, some machine learning-based methods are also considered the class imbalance problem by using different types of techniques, which include cost-sensitive learning (Jing *et al.* [71], Ryu *et al.* [79]), ensemble learning (Wang *et al.* [73, 81]) and data sampling algorithms (Wang *et al.* [74], Yang *et al.* [75], Chen *et al.* [76], Zhang *et al.* [82]). It can be seen that data sampling, ensemble learning, cost-sensitive learning and their hybrid methods are commonly used to deal with the class imbalance problem in the domain of defect prediction. Besides, most defect prediction studies use sampling techniques, possibly because they are simple, efficient and easy to be realised. For handling noises, noise-free data set is very important for building effective defect prediction models. Data contains noises, has a large impact on the defect prediction performance. Due to privacy-preserving issue, most data owners are not willing to share their data, which will restrict us to utilise these data for cross-project defect prediction, especially for new projects or projects without sufficient historical defect data. Hence, it is very necessary to further thoroughly investigate the dataset quality for future defect prediction studies.

5 Effort-aware context-based defect prediction studies

To prioritise quality assurance efforts, defect prediction techniques are often used to prioritise software instances based on their probability of having a defect or the number of defects [61, 117, 118]. With these defect prediction techniques, software engineers can allocate limited testing or inspection resources to the most defect-prone instances. Recently, effort-aware context-based defect prediction methods [117, 118] have been proposed. These methods

Table 8 Comparison of the defect prediction studies that handing dataset quality

Category	Study	Topic	Technique	Dataset	Year
class imbalance	Jing *et al.* [103]	method to solve class imbalance problem	ISDA	SOFTLAB, NASA, ReLink, AEEEM	2017
	Tan *et al.* [104]	online change prediction with different sampling techniques	simple duplicate, SMOTE, spread subsample, resampling with/without replacements	six open source projects	2015
	Ryu *et al.* [69]	method to solve cross-project class imbalance	boosting, under-sampling, over-sampling	NASA, SOFTLAB	2016
	Chen *et al.* [105]	method to solve class imbalance problem	ensemble random under-sampling	NASA	2016
	Wu *et al.* [106]	method to solve class imbalance problem	cost-sensitive learning	NASA	2016
	Liu *et al.* [107]	method to solve class imbalance problem	cost-sensitive learning	NASA	2014
	Rodriguze *et al.* [108]	predictive ability comparison of various class imbalance techniques	sampling, cost-sensitive, ensemble, hybrid	NASA	2014
	Malhotra and Khamna [109]	predictive ability comparison of various class imbalance techniques	resampling with replacement, spread subsample, SMOTE, MetaCost	six open source projects	2017
data noise	Tantithamthavorn *et al.* [112]	validation of mislabelling on the performance and interpretation	bootstrap resampling, random forest, Scott-Knott test	NASA, PROMISE	2015
	Herzig *et al.* [113]	validation of tangled code change on the performance	heuristic-based untangling, Caret package	five open source projects	2016
privacy-preserving data sharing	Peters *et al.* [116]	privacy-preserving algorithm for cross-project prediction	CLIFF, LeaF, MORPH	PROMISE	2015

factor in the effort needed to inspect or test code when evaluating the effectiveness of prediction models, leading to more realistic performance evaluations. Generally, effort-aware studies provide new interpretation and the practical adoption-oriented view of defect prediction results. These methods often use the LOC metric as a proxy measure for inspection effort. Considering the practical significance, more and more defect prediction studies make use of the effort-aware performance evaluation [21, 38, 52, 60, 68].

Zhou *et al.* [119] provided a comprehensive investigation of the class size on the effect between OO metrics and defect-proneness in the effort-aware prediction context. They employed statistical regression techniques to empirically study the effectiveness of defect prediction models. Results show that the performance of learned models can usually have significant improvement after removing the confounding effect in terms of both ranking and classification on effort-aware evaluation.

To perform an in-depth evaluation on the ability of slice-based cohesion metrics in effort-aware post-release defect prediction, Yang *et al.* [120] compared and evaluated the effect of slice-based cohesion metrics and baseline (code and process) metrics. They found that slice-based cohesion metrics are complementary to the code and process metrics. It suggests that there is practical value to apply slice-based cohesion metrics for the effort-aware post-release defect prediction.

To examine the effect of some package-modularisation metrics proposed by Sarkar *et al.* [121] in the context of effort-aware defect prediction, Zhao *et al.* [122] compared and evaluated the effectiveness of these new package-modularisation metrics and traditional package-level metrics. They found that these new package-modularisation metrics are useful for developing quality software systems in the effort-aware context.

To investigate the predictive effectiveness of simple unsupervised models in the context of effort-aware JIT prediction, Yang *et al.* [123] performed an empirical study to compare their unsupervised models with the state-of-the-art supervised models under three prediction settings including cross-validation, time-wise-cross-validation and cross-project prediction. They found that many simple unsupervised models achieve higher performance than the state-of-the-art supervised models in effort-aware JIT prediction.

To study the relationships between dependence clusters and defect-proneness in effort-aware defect prediction, Yang *et al.* [124] empirically evaluated the effect of dependence clusters with different statistical techniques. They found that large dependence clusters, functions inside dependence clusters tend to be more defect-prone, which help us to better understand dependence clusters and its effect on software quality in the effort-aware context.

To examine the predictive power of network measures in the effort-aware defect prediction, Ma *et al.* [125] performed an in-depth evaluation on the network measures with logistic regression technique. They found that it is practically useful to use network measures for effort-aware defect prediction. They also suggest that researchers should carefully decide whether and when to use network measures for defect prediction in practice.

Zhao *et al.* [126] provided a thorough study on the actual usefulness of client usage context in package cohesion for the effort-aware defect prediction. They evaluated the predictive ability of both context-based and non-context-based cohesion metrics taken alone or together for defect prediction, and found that it is practical value in package cohesion by considering client usage context for effort-aware defect prediction.

To examine the effect of sampling techniques on the performance in effort-aware defect prediction, Bennin *et al.* [127] took into account the testing effort with an effort-aware measure. They used four sampling techniques including random under-sampling, random over-sampling, SMOTE and purely SMOTE sampling for the experiments. Results show that the over-sampling techniques achieved higher performance than the under-sampling techniques.

Bennin *et al.* [10] leveraged the commonly used statistical and machine learning techniques to build 11 effort-aware prediction models for the more practical cross-release validation. They found that the performance of the prediction models is significantly dependent on the data set size and the percentage of defective instances in the data set.

Panichella *et al.* [128] presented to train defect predictors with the maximisation of their cost-effectiveness through GAs. They employed regression tree and generalised linear model to build defect prediction models, and found that regression models trained by GAs achieve better performance than their traditional counterparts.

Summary: Table 9 summarises the details of the effort-aware defect prediction studies. It can be seen that most works are based on the evaluation and analysis of the relationship between software metrics and defect-proneness, and prediction models constructed. Effort-aware context-based defect prediction studies consider the effort that is required to perform quality assurance activities like testing or code review on more prone to defective modules. Based on the prediction results, software engineers can allocate their limited testing resources to the defect-prone modules with the aim of finding more defects by using smaller effort. From the view of practice, it is more realistic and useful to apply effort-aware defect prediction models in the actual software development, which can improve the production efficiency and quality, reduce the development cost and software risk.

6 Empirical studies

In recent years, there have been many empirical studies to analyse and evaluate defect prediction methods from different aspects [31, 56, 58, 59]. Related ideas and brief descriptions of these papers are discussed and summarised as follows.

Shepperd *et al.* [129] investigated the extent to which the research group that performs a defect prediction study associates with the reported performance of defect prediction models. By conducting a meta-analysis of 42 primary studies, they found that the reported performance of a defect prediction model shares a strong relationship with the group of researchers who construct the models. Their findings suggest that many published defect prediction works are biased.

Recently, Tantithamthavorn *et al.* [130] conduct an alternative investigation of Shepperd *et al.*'s data [129]. They found that research group shares a strong association with other explanatory variables (dataset and metric families), the strong association among these explanatory variables makes it difficult to discern the impact of the research group on model performance, and the research group has a smaller impact than the metric family after mitigating the impact of this strong association. Their findings suggest that researchers experiment with a broader selection of datasets and metrics to combat any potential bias in the results.

Ghotra *et al.* [131] replicated prior study [58] to examine whether the impact of classification techniques on the performance of defect prediction models is significant or not in two experimental settings. They found that the performance with different classification techniques has small difference on the original NASA dataset which is consistent with the results of prior study. However, it has significantly different performance on the cleaned NASA dataset. Their results suggest that some classification techniques outperform others for defect prediction.

To investigate the effect of parameter on the performance of defect prediction models, Tantithamthavorn *et al.* [132] conducted an empirical study with an automated parameter optimisation tool, Caret. By evaluating candidate parameter settings, Caret can get the optimised setting based on the highest prediction performance. Results show that parameter settings can indeed significantly influence the performance of defect prediction models. The finding suggests that researchers should select right parameters of the classification techniques for the future defect prediction experiment.

To examine the bias and variance of model validation techniques in the field of defect prediction, Tantithamthavorn *et al.* [70] conducted an in-depth study with 12 most widely used model validation techniques for the evaluation. They found that single-repetition holdout validation tends to yield estimates with 46–229% more bias and 53–863% more variance than the top-ranked

model validation techniques, while out-of-sample bootstrap validation achieves the best balance between the bias and variance of estimates. Hence, they suggested that researchers should adopt out-of-sample bootstrap validation in future defect prediction studies.

To recognise classification techniques which perform well in software defect prediction, Bowes *et al.* [133] applied four classifiers including random forest, naive Bayes, RPart and SVM to predict and analyse the level of prediction uncertainty through the investigation of individual defects. They found that the predictive performance of four classifiers is similar, but each detects different sets of defects. Based on the experimental results, they concluded that the prediction performance of classifiers with ensemble decision-making strategies outperforms majority voting.

Due to lack of an identical performance evaluation measure to directly compare different defect prediction models, Bowes *et al.* [134] developed a Java tool 'DConfusion' that allows researchers and practitioners to transform many performance measures back into a confusion matrix. They found that the tool can produce very small errors by using a variety of datasets and learners for re-computing the confusion matrix.

To examine the effectiveness of search-based techniques and their hybridised versions for defect prediction, Malhotra and Khanna [135] performed an empirical comparison of five search-based, five hybridised, four machine learning techniques and a statistical technique. Through comparing the predictive performance of each model, they advocated that researchers should employ hybridised techniques to develop defect prediction models for recognising software defects.

Caglayan *et al.* [47] replicated a prior study [45] to examine the merits of organisational metrics on the performance of defect prediction models for large-scale enterprise software. They separately extracted organisational, code complexity, code churn and pre-release defect metrics to build prediction models. They found that model with organisational metrics outperforms than both churn metric and pre-release metric models.

To investigate how different aggregation schemes have effect on defect prediction models, Zhang *et al.* [136] conducted an in-depth analysis of 11 aggregation schemes on 255 open source software projects. They found that aggregation has large effect on both correlation among metrics and the correlation between metrics and defects. Using only the summation does not often achieve the best performance, and it tends to underestimate the performance of defect prediction models. Given their findings, they advised researchers to explore various aggregation schemes in future defect prediction studies.

To validate the feasibility of using a simplified metric set for defect prediction, He *et al.* [137] constructed three types of predictors in three scenarios with six typical classifiers. They found that it is viable and practical to use a simplified metric set for defect prediction, and the built models with the simplified metric subset can provide satisfactory prediction performance.

Yang *et al.* [138] conducted an empirical study to examine the relationship between end-slice-based and metric-slice-based cohesion metrics as well as compared the predictive power of different type of slice for slice-based cohesion metrics. They found that end-slice-based and metric-slice-based cohesion metrics have little difference. In practice, they suggested selecting end slice for computing slice-based cohesion metrics in defect prediction.

Jaafar *et al.* [139] conducted an empirical evaluation on the impact of design pattern and anti-pattern dependencies on changes and faults in OO systems. They found that classes of anti-patterns dependencies are more prone to defect than others while classes of design patterns dependencies are not hold true. They also observed that classes having dependencies with anti-patterns depend on the structural changes which are the most common changes.

Chen *et al.* [140] conducted an empirical study to examine the predictive effectiveness of network measures in high severity defect prediction. They separately used logistic regression to analysis the relationships between each network measure and defect-proneness, and then evaluate their predictive powers compared to CMs. They found that network measures are of practical value in high severity defect prediction.

To investigate the relationship between process metrics and the number of defects, Madeyski and Jureczko [22] provided an in-depth evaluation which process metrics have large effect on improving the predictors compared to product metrics. They found that the number of distinct committers (NDC) metric can significantly improve the performance of prediction models and they recommended to use the NDC process metric for future defect prediction studies.

To investigate the performance of JIT models in the context of cross-project defect prediction, Kamei *et al.* [141] conducted an empirical evaluation on 11 open source projects. They found that training and test projects with similar data distribution usually can obtain better performance, and combining the data of multiple projects or several prediction models tend to produce strong performance. For the JIT cross-project prediction models with the

Table 9 Comparison of effort-aware defect prediction studies

Study	Topic	Technique	Data set	Year
Zhou *et al.* [119]	investigation of confounding effect of class size	Sobel's statistical test, linear regression	six open source Java projects	2014
Yang *et al.* [120]	predictive ability analysis of slice-based cohesion metrics for post-release defects	principal component analysis, univariate/multivariate logistic regression	five open source C project	2015
Zhao *et al.* [122]	predictive ability evaluation of package-modularisation metrics	univariate/multivariate logistic regression	6 open source Java projects	2015
Yang *et al.* [123]	predictive ability comparison of simple unsupervised models for JIT prediction	change metrics-based unsupervised techniques	JIT	2016
Yang *et al.* [124]	validation of the relationship between dependence clusters and defect-proneness at the function level	cluster, Spearman's rank correlation, Fisher's exact test, univariate/ multivariate logistic regression	five open source C projects	2016
Ma *et al.* [125]	a comprehensive evaluation on the predictive effectiveness of network measures	univariate/multivariate logistic regression	PROMISE	2016
Zhao *et al.* [126]	validation on the value of considering client usage context in package cohesion	principal component analysis, univariate/multivariate logistic regression	ten open source Jave projects	2017
Bennin *et al.* [127]	predictive ability comparison of sampling techniques	sampling, statistical and machine learning	ten open source projects	2016
Bennin *et al.* [10]	predictive ability comparison in cross-release prediction	statistical and machine learning	25 open source projects	2016
Panichella *et al.* [128]	fit ability comparison of search-based algorithm	GA, regression tree, generalised liner model	PROMISE	2016

utilisation of carefully selected data, these models usually tend to perform best.

To construct a universal defect prediction model, Zhang *et al.* [67] presented a context-aware rank transformation method by studying six context factors. Through a study of 1385 open source projects, the universal model achieves comparable performance to the within-project models, produces similar performance on five external projects and performs similarly among projects with different context factors.

Petric *et al.* [142] conducted an empirical study to examine the use of an explicit diversity technique with stacking ensemble that whether defect prediction can be improved. They employed the stacking ensemble technique to combine four different types of classifiers and a weighted accuracy diversity model. Based on eight publicly available projects, the results demonstrate that stacking ensembles perform better than other defect prediction models and the essential factor is the use of diversity ensemble.

Liu *et al.* [143] performed an empirical study of two-stage data preprocessing method for software defect prediction, which includes feature selection and instance reduction two stages. The aim of feature selection is to remove irrelevant, redundant features, and the aim of instance reduction is to undersample non-defective instances. Experimental evaluation on eclipse and NASA projects shows the effectiveness of the proposed two-stage data preprocessing method.

Xu *et al.* [144] conducted an empirical evaluation on the impact of 32 feature selection techniques for defect prediction. They found that feature selection techniques are useful and the performance of predictors trained using these techniques exhibits significant difference over all the projects.

Summary: Table 10 shows the comparison of the empirical studies on defect prediction. We can observe that these papers mainly focus on different machine learning and statistical methods, classifiers or classifier ensemble techniques, parameter optimisation technique, model validation techniques, performance evaluation measures, various software metrics and feature selection techniques. The above empirical studies provide a variety of profound, original, valuable and practical insights. These findings can help software researchers and practitioners to better understand the results reported by the papers and give some practical guidelines in future defect prediction studies.

7 Future directions and challenges

Although numerous efforts have been made and many outstanding studies have been proposed in software defect prediction, there are still many potential future challenges, and some new open problems require further investigation and exploration. Here, we address some possible future research challenges on the defect prediction problem.

Table 10 Empirical studies on defect prediction

Study	Topic	Technique	Year
Shepperd *et al.* [129]	identification of the effect of model building factors on predictive performance	statistical leaning	2014
Tantithamthavorn *et al.* [130]	identification of the effect of model building factors on predictive performance	statistical leaning	2016
Ghotra *et al.* [131]	predictive ability comparison of classification techniques	31 statistical and machine learning algorithms	2015
Tantithamthavorn *et al.* [132]	predictive ability analysis of automated parameter optimisation technique	30 statistical and machine learning algorithms	2016
Tantithamthavorn *et al.* [70]	predictive ability comparison of model validation techniques	naive Bayes, logistic regression, random forest	2017
Bowes *et al.* [134]	applicability of a technique to allow cross study performance evaluation	rpart, random forest, naive Bayes	2014
Bowes *et al.* [133]	comparison the performance of classifiers on predicting defects	random forest, naive Bayes, rpart, support vector machine	2017
Malhortra and Khanna [135]	validation and comparison of search-based techniques and their hybridised versions	five search-based techniques, five hybridised techniques	2017
Caglayan *et al.* [47]	validation the merits of organisational metrics	correlation analysis, principal component analysis	2015
Zhang *et al.* [136]	evaluation and analysis the performance of different aggregation schemes	Spearman correlation, gain ratio, random forest, linear regression	2017
He *et al.* [137]	feasibility of a simplified metric set in different prediction scenarios	decision tree/table, naive Bayes, logistic regression, support vector machine, Bayesian network	2015
Yang *et al.* [138]	applicability of slice types on slice-based cohesion metrics	correlation analysis, principal component analysis	2016
Jaafar *et al.* [139]	evaluation of the impact of design pattern and anti-pattern dependencies on change and faults	Fisher's exact test, odds ratio	2016
Chen *et al.* [140]	utility of network measures in high severity prediction	univariate/multivariate logistic regression	2016
Madeyski and Jureczko [22]	predictive ability comparison of process metrics	stepwise linear regression	2015
Kamei *et al.* [141]	validation of JIT prediction in a cross-project context	random forest	2016
Zhang *et al.* [67]	a method to build and evaluate a universal defect prediction model	clustering, naive Bayes, logistic regression	2016
Petric *et al.* [142]	methods to build and evaluate diversity ensemble models	naive Bayes, decision tree, k nearest neighbour, support vector machine, stacking ensemble	2016
Liu *et al.* [143]	validation of data preprocessing techniques	information gain, chi-square, ReliefF, symmetic uncertainty, cosine similarity, random sampling	2016
Xu *et al.* [144]	predictive ability comparison of feature selection techniques	filter, wrapper, clustering, extraction-based algorithms	2016

(i) *Generalisation*: Most defect prediction studies are verified in open source software projects. The main reason is that these projects easily collect, archive practical development history and make their data publicly available. However, current defect prediction methods might not be generalisable to other closed software projects such as proprietary and commercial systems. That is, the lack of availability of such proprietary and commercial data leaves open the question of whether defect prediction models built using data from open source projects would apply to these projects. Such defect prediction methods are not investigated in depth. Thus, there need to make more partnership with business partners and have access to their data repositories.

(ii) *Overcoming imbalance*: Software defect prediction always suffers from class imbalance, namely the number of defective instances is often much less than the number of non-defective instances. This imbalanced data distribution hampers the effectiveness of many defect prediction models. Although some defect prediction studies [65, 69, 103, 104, 109] have been presented to deal with the class imbalance problem, it is still necessary and important to overcome this problem in the future work and further improve the performance of defect prediction.

(iii) *Focusing on effort*: It is very necessary and important to evaluate the prediction models in a realistic setting, e.g. how much effort can reduce for testing and code inspection with defect prediction models. Recent studies have attempted to address such problem when considering the effort [117, 118, 120, 123, 127]. These defect prediction studies usually utilise the lines of CM as a proxy for effort. In future effort-aware defect prediction models, researchers should examine what is the best way to measure effort and provide better practical guidelines. Such research can have a larger effect on the future applicability of defect prediction techniques in practice.

(iv) *Heterogeneous defect prediction*: Recently, several heterogeneous defect prediction models have been developed [66, 98, 100], which use heterogeneous metric data from other projects. It provides a new perspective to defect prediction. For new projects or projects lacking in sufficient historical data, it is very meaningful to study and use these methods for predicting defects. Due to the different data distributions exist among the source and target projects, it is difficult to build good defect prediction models that achieve the satisfactory performance. Furthermore, studies on heterogeneous cross-project prediction feasibility are not mature yet in practice. At this point, there remains as an open research area for practical use of heterogeneous cross-project defect prediction.

(v) *Privacy preservation issue*: Due to the business sensitivity and privacy concerns, most companies are not willing to share their data [64, 114–116]. In this scenario, it is often difficult to extract data from industrial companies. Therefore, current defect prediction models may not work for those of proprietary and industrial projects. If we have more available such datasets, building cross-project defect prediction models will be more sound. In the future, it is very important and necessary to extensively study the privacy issue in the context of cross-project defect prediction.

(vi) *Fair evaluation*: Up to now, numerous methods have been proposed to solve the defect prediction problem. However, the corresponding evaluations are inconsistent and inadequate, which may lead to inappropriate conclusions about the performance of defect prediction methods. Hence, more fair evaluation for defect prediction is very necessary.

(vii) *Reproducibility*: Some defect prediction papers [51, 52, 78, 83, 123] are providing replication packages such as dataset, implementation scripts, and additional settings file. The main goal of replication of defect prediction is not only to replicate and validity their experiments but also to allow other researchers to compare the prediction performance of new models with the original studies [14, 145]. Though having access to the replication packages of previous studies, there may not be get the exact same results. A selection of evaluation procedures, configuration parameters and performance metrics has a large impact on the reproducibility. However, we recommend that researchers could provide their complete replication packages in future defect prediction studies.

8 Conclusion

Software quality assurance has become one of the most significant and expensive phase during the development of high assurance software systems. As software systems play an increasingly key role in our daily lives, their complexity continues to increase. It is very difficult to make their quality assurance since the increased complexity of software systems. The defect prediction models can identify the defect-prone modules so that quality assurance teams can effectively allocate limited resources for testing and code inspection by putting more effort on the defect-prone modules.

In recent years, new defect prediction techniques, problems and applications are emerging quickly. This paper attempts to systematically summarise all typical works on software defect prediction in recent years. Based on the results obtained in this work, this paper will help researchers and software practitioners to better understand the previous defect prediction studies from data sets, software metrics, evaluation measures, and modelling techniques perspectives in an easy and effective way.

9 Acknowledgments

The authors thank the editors and anonymous reviewers for their constructive comments and suggestions. The authors also thank Professor Jifeng Xuan and Professor Xiaoyuan Xie from School of Computer at Wuhan University for their insightful advice. This work was supported by the NSFC-Key Project of General Technology Fundamental Research United Fund under grant no. U1736211, the National Key Research and Development Program of China under grant no. 2017YFB0202001, the National Nature Science Foundation of China under grant nos. 61672208, 41571417 and U1504611, the Science and Technique Development Program of Henan under grant no. 172102210186, and the Research Foundation of Henan University under grant no. 2015YBZR024.

10 References

[1] Hall, T., Beecham, S., Bowes, D., *et al.*: 'A systematic literature review on fault prediction performance in software engineering', *IEEE Trans. Softw. Eng.*, 2012, **38**, (6), pp. 1276–1304

[2] Menzies, T., Milton, Z., Turhan, B., *et al.*: 'Defect prediction from static code features: current results, limitations, new approaches', *Autom. Softw. Eng.*, 2010, **17**, (4), pp. 375–407

[3] Catal, C., Diri, B.: 'A systematic review of software fault prediction studies', *Expert Syst. Appl.*, 2009, **36**, (4), pp. 7346–7354

[4] Catal, C.: 'Software fault prediction: a literature review and current trends', *Expert Syst. Appl.*, 2011, **38**, (4), pp. 4626–4636

[5] Malhotra, R.: 'A systematic review of machine learning techniques for software fault prediction', *Appl. Soft Comput.*, 2015, **27**, pp. 504–518

[6] Naik, K., Tripathy, P.: '*Software testing and quality assurance: theory and practice*' (John Wiley & Sons, Hoboken, NJ, 2011)

[7] Menzies, T., Greenwald, J., Frank, A.: 'Data mining static code attributes to learn defect predictors', *IEEE Trans. Softw. Eng.*, 2007, **33**, (1), pp. 2–13

[8] Song, Q., Jia, Z., Shepperd, M., *et al.*: 'A general software defect proneness prediction framework', *IEEE Trans. Softw. Eng.*, 2011, **37**, (3), pp. 356–370

[9] Herzig, K.: 'Using pre-release test failures to build early post-release defect prediction models'. Proc. IEEE 25th Int. Symp. Software Reliability Engineering, 2014, pp. 300–311

[10] Bennin, K.E., Toda, K., Kamei, Y., *et al.*: 'Empirical evaluation of cross-release effort-aware defect prediction models'. Proc. IEEE Int. Conf. Software Quality, Reliability and Security, 2016, pp. 214–221

[11] Zimmermann, T., Nagappan, N., Gall, H., *et al.*: 'Cross-project defect prediction: a large scale experiment on data vs. domain vs. process'. Proc. 7th joint Meeting of the European Software Engineering Conf. ACM SIGSOFT Int. Symp. Foundations of Software Engineering, 2009, pp. 91–100

[12] Turhan, B., Menzies, T., Bener, A.B., *et al.*: 'On the relative value of cross-company and within-company data for defect prediction', *Empir. Softw. Eng.*, 2009, **14**, (5), pp. 540–578

[13] He, Z., Shu, F., Yang, Y., *et al.*: 'An investigation on the feasibility of cross-project defect prediction', *Autom. Softw. Eng.*, 2012, **19**, (2), pp. 167–199

[14] Kamei, Y., Shihab, E.: 'Defect prediction: accomplishments and future challenges'. Proc. IEEE 23rd Int. Conf. Software Analysis, Evolution, and Reengineering, 2016, pp. 33–45

[15] Kim, S., Whitehead, E.J., Zhang, Y.: 'Classifying software changes: clean or buggy?' *IEEE Trans. Softw. Eng.*, 2008, **34**, (2), pp. 181–196

[16] Nam, J., Pan, S.J., Kim, S.: 'Transfer defect learning'. Proc. 35th Int. Conf. Software Engineering, 2013, pp. 382–391

[17] Turhan, B., Mısırlı, A.T., Bener, A.: 'Empirical evaluation of the effects of mixed project data on learning defect predictors', *Inf. Softw. Technol.*, 2013, **55**, (6), pp. 1101–1118

[18] Zhang, Y., Lo, D., Xia, X., *et al.*: 'An empirical study of classifier combination for cross-project defect prediction'. Proc. IEEE 39th Annual Computer Software and Applications Conf., 2015, pp. 264–269

[19] Krishna, R., Menzies, T., Fu, W.: 'Too much automation? The bellwether effect and its implications for transfer learning'. Proc. 31st Int. Conf. Automated Software Engineering, 2016, pp. 122–131

[20] Andreou, A.S., Chatzis, S.P.: 'Software defect prediction using doubly stochastic Poisson processes driven by stochastic belief networks', *J. Syst. Softw.*, 2016, **122**, pp. 72–82

[21] Rahman, F., Devanbu, P.: 'How, and why, process metrics are better'. Proc. 2013 Int. Conf. Software Engineering, 2013, pp. 432–441

[22] Madeyski, L., Jureczko, M.: 'Which process metrics can significantly improve defect prediction models? An empirical study', *Softw. Qual. J.*, 2015, **23**, (3), pp. 393–422

[23] Radjenović, D., Heričko, M., Torkar, R., *et al.*: 'Software fault prediction metrics: a systematic literature review', *Inf. Softw. Technol.*, 2013, **55**, (8), pp. 1397–1418

[24] Halstead, M.H.: '*Elements of software science*'. vol. **7** (Elsevier, New York, 1977)

[25] McCabe, T.J.: 'A complexity measure', *IEEE Trans. Softw. Eng.*, 1976, **2**, (4), pp. 308–320

[26] Chidamber, S.R., Kemerer, C.F.: 'A metrics suite for object oriented design', *IEEE Trans. Softw. Eng.*, 1994, **20**, (6), pp. 476–493

[27] Abreu, F.B., Carapuça, R.: 'Candidate metrics for object-oriented software within a taxonomy framework', *J. Syst. Softw.*, 1994, **26**, (1), pp. 87–96

[28] Nagappan, N., Ball, T.: 'Use of relative code churn measures to predict system defect density'. Proc. 27th Int. Conf. Software Engineering, 2005, pp. 284–292

[29] Moser, R., Pedrycz, W., Succi, G.: 'A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction'. Proc. 30th Int. Conf. Software Engineering, 2008, pp. 181–190

[30] Hassan, A.E.: 'Predicting faults using the complexity of code changes'. Proc. 31st Int. Conf. Software Engineering, 2009, pp. 78–88

[31] D'Ambros, M., Lanza, M., Robbes, R.: 'Evaluating defect prediction approaches: a benchmark and an extensive comparison', *Empir. Softw. Eng.*, 2012, **17**, (4–5), pp. 531–577

[32] Weyuker, E.J., Ostrand, T.J., Bell, R.M.: 'Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models', *Empir. Softw. Eng.*, 2008, **13**, (5), pp. 539–559

[33] Pinzger, M., Nagappan, N., Murphy, B.: 'Can developer-module networks predict failures?' Proc. 16th ACM SIGSOFT Int. Symp. Foundations of Software Engineering, 2008, pp. 2–12

[34] Meneely, A., Williams, L., Snipes, W., *et al.*: 'Predicting failures with developer networks and social network analysis'. Proc. 16th ACM SIGSOFT Int. Symp. Foundations of Software Engineering, 2008, pp. 13–23

[35] Bird, C., Nagappan, N., Murphy, B., *et al.*: 'Don't touch my code!: examining the effects of ownership on software quality'. Proc. 19th ACM SIGSOFT Symp. and the 13th European Conf. Foundations of Software Engineering, 2011, pp. 4–14

[36] Rahman, F.: 'Ownership, experience and defects: a fine-grained study of authorship'. Proc. 33rd Int. Conf. Software Engineering, 2011, pp. 491–500

[37] Posnett, D., Devanbu, P., Filkov, V.: 'Dual ecological measures of focus in software development'. Proc. 35th Int. Conf. Software Engineering, 2013, pp. 452–461

[38] Jiang, T., Tan, L., Kim, S.: 'Personalized defect prediction'. Proc. IEEE/ACM 28th Int. Conf. Automated Software Engineering, 2013, pp. 279–289

[39] Lee, T., Nam, J., Han, D., *et al.*: 'Developer micro interaction metrics for software defect prediction', *IEEE Trans. Softw. Eng.*, 2016, **42**, (11), pp. 1015–1035

[40] Zimmermann, T., Nagappan, N.: 'Predicting defects using network analysis on dependency graphs'. Proc. 30th Int. Conf. Software Engineering, 2008, pp. 531–540

[41] Bird, C., Nagappan, N., Gall, H., *et al.*: 'Using socio-technical networks to predict failures', Proc. 20th IEEE Int. Symp. Software Reliability Engineering, 2009

[42] D'Ambros, M., Lanza, M., Robbes, R.: 'On the relationship between change coupling and software defects'. Proc. 16th Working Conf. Reverse Engineering, 2009, pp. 135–144

[43] Hu, W., Wong, K.: 'Using citation influence to predict software defects'. Proc. 10th Working Conf. Mining Software Repositories, 2013, pp. 419–428

[44] Herzig, K., Just, S., Rau, A., *et al.*: 'Predicting defects using change genealogies'. Proc. IEEE 24th Int. Symp. Software Reliability Engineering, 2013, pp. 118–127

[45] Nagappan, N., Murphy, B., Basili, V.: 'The influence of organizational structure on software quality'. ACM/IEEE 30th Int. Conf. Software Engineering, 2008, pp. 521–530

[46] Mockus, A.: 'Organizational volatility and its effects on software defects'. Proc. Eighteenth ACM SIGSOFT Int. Symp. Foundations of Software Engineering, 2010, pp. 117–126

[47] Caglayan, B., Turhan, B., Bener, A., *et al.*: 'Merits of organizational metrics in defect prediction: an industrial replication'. Proc. IEEE/ACM 37th IEEE Int. Conf. Software Engineering, 2015, pp. 89–98

[48] Bacchelli, A., D'Ambros, M., Lanza, M.: 'Are popular classes more defect prone?' Proc. 13th Int. Conf. Fundamental Approaches to Software Engineering, 2010, pp. 59–73

[49] Taba, S.E.S., Khomh, F., Zou, Y., *et al.*: 'Predicting bugs using antipatterns'. Proc. 29th Int. Conf. Software Maintenance, 2013, pp. 270–279

[50] Zhang, H.: 'An investigation of the relationships between lines of code and defects'. Proc. IEEE Int. Conf. Software Maintenance, 2009, pp. 274–283

[51] Wu, R., Zhang, H., Kim, S., *et al.*: 'Relink: recovering links between bugs and changes'. Proc. 19th ACM SIGSOFT Symp. the Foundations of Software Engineering and 13th European Software Engineering Conf., 2011, pp. 15–25

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

[53] Zimmermann, T., Premraj, R., Zeller, A.: 'Predicting defects for eclipse'. Proc. Third Int. Workshop on Predictor Models in Software Engineering, 2007, pp. 9–15

[54] Kim, S., Zhang, H., Wu, R., *et al.*: 'Dealing with noise in defect prediction'. Proc. 33rd Int. Conf. Software Engineering, 2011, pp. 481–490

[55] Altinger, H., Siegl, S., Dajsuren, Y., *et al.*: 'A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software'. Proc. 12th Working Conf. Mining Software Repositories, 2015, pp. 494–497

[56] Shepperd, M., Song, Q., Sun, Z., *et al.*: 'Data quality: some comments on the NASA software defect datasets', *IEEE Trans. Softw. Eng.*, 2013, **39**, (9), pp. 1208–1215

[57] Jureczko, M., Madeyski, L.: 'Towards identifying software project clusters with regard to defect prediction'. Proc. 6th Int. Conf. Predictive Models in Software Engineering, 2010, pp. 1–10

[58] 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.*, 2008, **34**, (4), pp. 485–496

[59] Jiang, Y., Cukic, B., Ma, Y.: 'Techniques for evaluating fault prediction models', *Empir. Softw. Eng.*, 2008, **13**, (5), pp. 561–595

[60] Mende, T., Koschke, R.: 'Revisiting the evaluation of defect prediction models'. Proc. 5th Int. Conf. Predictor Models in Software Engineering, 2009, pp. 1–10

[61] Arisholm, E., Briand, L.C., Johannessen, E.B.: 'A systematic and comprehensive investigation of methods to build and evaluate fault prediction models', *J. Syst. Softw.*, 2010, **83**, (1), pp. 2–17

[62] Xiao, X., Lo, D., Xin, X., *et al.*: 'Evaluating defect prediction approaches using a massive set of metrics: an empirical study'. Proc. 30th Annual ACM Symp. Applied Computing, 2015, pp. 1644–1647

[63] Menzies, T., Dekhtyar, A., Distefano, J., *et al.*: 'Problems with precision: a response to 'comments on 'data mining static code attributes to learn defect predictors'", *IEEE Trans. Softw. Eng.*, 2007, **33**, (9), pp. 635–636

[64] Peters, F., Menzies, T., Gong, L., *et al.*: 'Balancing privacy and utility in cross-company defect prediction', *IEEE Trans. Softw. Eng.*, 2013, **39**, (8), pp. 1054–1068

[65] Wang, S., Yao, X.: 'Using class imbalance learning for software defect prediction', *IEEE Trans. Reliab.*, 2013, **62**, (2), pp. 434–443

[66] Jing, X.Y., Wu, F., Dong, X., *et al.*: 'Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning'. Proc. 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 496–507

[67] Zhang, F., Mockus, A., Keivanloo, I., *et al.*: 'Towards building a universal defect prediction model with rank transformed predictors', *Empir. Softw. Eng.*, 2016, **21**, (5), pp. 1–39

[68] Rahman, F., Posnett, D., Devanbu, P.: 'Recalling the imprecision of cross-project defect prediction'. Proc. ACM SIGSOFT 20th Int. Symp. the Foundations of Software Engineering, 2012, pp. 1–11

[69] Ryu, D., Choi, O., Baik, J.: 'Value-cognitive boosting with a support vector machine for cross-project defect prediction', *Empir. Softw. Eng.*, 2016, **21**, (1), pp. 43–71

[70] Tantithamthavorn, C., Mcintosh, S., Hassan, A., *et al.*: 'An empirical comparison of model validation techniques for defect prediction models', *IEEE Trans. Softw. Eng.*, 2017, **43**, (1), pp. 1–18

[71] Jing, X.Y., Ying, S., Zhang, Z.W., *et al.*: 'Dictionary learning based software defect prediction'. Proc. 36th Int. Conf. Software Engineering, 2014, pp. 414–423

[72] Jing, X.Y., Zhang, Z.W., Ying, S., *et al.*: 'Software defect prediction based on collaborative representation classification'. Companion Proc. 36th Int. Conf. Software Engineering, 2014, pp. 632–633

[73] Wang, T., Zhang, Z., Jing, X.Y., *et al.*: 'Multiple kernel ensemble learning for software defect prediction', *Autom. Softw. Eng.*, 2016, **23**, (4), pp. 569–590

[74] Wang, S., Liu, T., Tan, L.: 'Automatically learning semantic features for defect prediction'. Proc. 38th Int. Conf. Software Engineering, 2016, pp. 297–308

[75] Yang, X., Lo, D., Xia, X., *et al.*: 'Deep learning for just-in-time defect prediction'. Proc. IEEE Int. Conf. Software Quality, Reliability and Security, 2015, pp. 17–26

[76] Chen, L., Fang, B., Shang, Z., *et al.*: 'Negative samples reduction in cross-company software defects prediction', *Inf. Softw. Technol.*, 2015, **62**, pp. 67–77

[77] Xia, X., Lo, D., Pan, S.J., *et al.*: 'Hydra: massively compositional model for cross-project defect prediction', *IEEE Trans. Softw. Eng.*, 2016, **42**, (10), pp. 977–998

[78] Canfora, G., Lucia, A.D., Penta, M.D., *et al.*: 'Defect prediction as a multiobjective optimization problem', *Softw. Test. Verif. Reliab.*, 2015, **25**, (4), pp. 426–459

[79] Ryu, D., Jang, J.I., Baik, J.: 'A transfer cost-sensitive boosting approach for cross-project defect prediction', *Softw. Qual. J.*, 2017, **25**, (1), pp. 235–272

[80] Yang, X., Lo, D., Xia, X., *et al.*: 'Tlel: a two-layer ensemble learning approach for just-in-time defect prediction', *Inf. Softw. Technol.*, 2017, **87**, pp. 206–220

[81] Wang, T., Zhang, Z., Jing, X.Y., *et al.*: 'Non-negative sparse-based semiboost for software defect prediction', *Softw. Test. Verif. Reliab.*, 2016, **26**, (7), pp. 498–515

[82] Zhang, Z., Jing, X.Y., Wang, T.: 'Label propagation based semi-supervised learning for software defect prediction', *Autom. Softw. Eng.*, 2017, **24**, (1), pp. 47–69

[83] Nam, J., Kim, S.: 'Clami: defect prediction on unlabeled datasets'. Proc. 30th IEEE/ACM Int. Conf. Automated Software Engineering, 2015, pp. 1–12

[84] Zhang, F., Zheng, Q., Zou, Y., *et al.*: 'Cross-project defect prediction using a connectivity-based unsupervised classifier'. Proc. 38th Int. Conf. Software Engineering, 2016, pp. 309–320

[85] Okutan, A., Yıldız, O.T.: 'Software defect prediction using Bayesian networks', *Empir. Softw. Eng.*, 2014, **19**, (1), pp. 154–181

[86] Bowes, D., Hall, T., Harman, M., *et al.*: 'Mutation-aware fault prediction'. Proc. 25th Int. Symp. Software Testing and Analysis, 2016, pp. 330–341

[87] Chen, T.H., Shang, W., Nagappan, M., *et al.*: 'Topic-based software defect explanation', *J. Syst. Softw.*, 2017, **129**, pp. 79–106

[88] Shivaji, S., Whitehead, E.J., Akella, R., *et al.*: 'Reducing features to improve code change-based bug prediction', *IEEE Trans. Softw. Eng.*, 2013, **39**, (4), pp. 552–569

[89] Gao, K., Khoshgoftaar, T.M., Wang, H., *et al.*: 'Choosing software metrics for defect prediction: an investigation on feature selection techniques', *Softw. Pract. Experience*, 2011, **41**, (5), pp. 579–606

[90] Laradji, I.H., Alshayeb, M., Ghouti, L.: 'Software defect prediction using ensemble learning on selected features', *Inf. Softw. Technol.*, 2015, **58**, pp. 388–402

[91] Liu, S., Chen, X., Liu, W., *et al.*: 'Fecar: A feature selection framework for software defect prediction'. Proc. IEEE 38th Annual Computer Software and Applications Conf., 2014, pp. 426–435

[92] Liu, W., Liu, S., Gu, Q., *et al.*: 'Fecs: a cluster based feature selection method for software fault prediction with noises'. Proc. IEEE 39th Annual Computer Software and Applications Conf., 2015, pp. 276–281

[93] Xu, Z., Xuan, J., Liu, J., *et al.*: 'Michac: defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering'. Proc. IEEE 23rd Int. Conf. Software Analysis, Evolution, and Reengineering, 2016, pp. 370–381

[94] Menzies, T., Butcher, A., Cok, D., *et al.*: 'Local versus global lessons for defect prediction and effort estimation', *IEEE Trans. Softw. Eng.*, 2013, **39**, (6), pp. 822–834

[95] Bettenburg, N., Nagappan, M., Hassan, A.E.: 'Towards improving statistical modeling of software engineering data: think locally, act globally!', *Empir. Softw. Eng.*, 2015, **20**, (2), pp. 294–335

[96] Herbold, S., Trautsch, A., Grabowski, J.: 'Global vs. local models for cross-project defect prediction', *Empir. Softw. Eng.*, 2017, **22**, (4), pp. 1866–1902

[97] Mezouar, M.E., Zhang, F., Zou, Y.: 'Local versus global models for effort-aware defect prediction'. Proc. 26th Annual Int. Conf. Computer Science and Software Engineering, 2016, pp. 178–187

[98] Nam, J., Kim, S.: 'Heterogeneous defect prediction'. Proc. 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 508–519

[99] He, P., Li, B., Ma, Y.: 'Towards cross-project defect prediction with imbalanced feature sets', CoRR, 2014, abs/1411.4228. Available at http://arxiv.org/abs/1411.4228

[100] Cheng, M., Wu, G., Jiang, M., *et al.*: 'Heterogeneous defect prediction via exploiting correlation subspace'. Proc. 28th Int. Conf. Software Engineering and Knowledge Engineering, 2016, pp. 171–176

[101] Zhang, H., Zhang, X.: 'Comments on 'data mining static code attributes to learn defect predictors'', *IEEE Trans. Softw. Eng.*, 2007, **33**, (9), pp. 635–637

[102] He, H., Garcia, E.A.: 'Learning from imbalanced data', *IEEE Trans. Knowl. Data Eng.*, 2009, **21**, (9), pp. 1263–1284

[103] Jing, X.Y., Wu, F., Dong, X., *et al.*: 'An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems', *IEEE Trans. Softw. Eng.*, 2017, **43**, (4), pp. 321–339

[104] Tan, M., Tan, L., Dara, S., *et al.*: 'Online defect prediction for imbalanced data'. Proc. 37th Int. Conf. Software Engineering, 2015, pp. 99–108

[105] Chen, L., Fang, B., Shang, Z., *et al.*: 'Tackling class overlap and imbalance problems in software defect prediction', *Softw. Qual. J.*, 2018, **26**, (1), pp. 97–125

[106] Wu, F., Jing, X.Y., Dong, X., *et al.*: 'Cost-sensitive local collaborative representation for software defect prediction'. Proc. Int. Conf. Software Analysis, Testing and Evolution, 2016, pp. 102–107

[107] Liu, M., Miao, L., Zhang, D.: 'Two-stage cost-sensitive learning for software defect prediction', *IEEE Trans. Reliab.*, 2014, **63**, (2), pp. 676–686

[108] Rodriguez, D., Herraiz, I., Harrison, R., *et al.*: 'Preliminary comparison of techniques for dealing with imbalance in software defect prediction'. Proc. 18th Int. Conf. Evaluation and Assessment in Software Engineering, 2014, pp. 1–10

[109] Malhotra, R., Khanna, M.: 'An empirical study for software change prediction using imbalanced data', *Empir. Softw. Eng.*, 2017, **22**, (6), pp. 2806–2851

[110] Herzig, K., Just, S., Zeller, A.: 'It's not a bug, it's a feature: how misclassification impacts bug prediction'. Proc. 2013 Int. Conf. Software Engineering, 2013, pp. 392–401

[111] Rahman, F., Posnett, D., Herraiz, I., *et al.*: 'Sample size vs. Bias in defect prediction'. Proc. 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 147–157

[112] Tantithamthavorn, C., McIntosh, S., Hassan, A.E., *et al.*: 'The impact of mislabelling on the performance and interpretation of defect prediction models'. Proc. 37th Int. Conf. Software Engineering, 2015, pp. 812–823

[113] Herzig, K., Just, S., Zeller, A.: 'The impact of tangled code changes on defect prediction models', *Empir. Softw. Eng.*, 2016, **21**, (2), pp. 303–336

[114] Peters, F., Menzies, T.: 'Privacy and utility for defect prediction: experiments with morph'. Proc. 34th Int. Conf. Software Engineering, 2012, pp. 189–199

[115] Qi, F., Jing, X.Y., Zhu, X., *et al.*: 'Privacy preserving via interval covering based subclass division and manifold learning based bi-directional obfuscation for effort estimation'. Proc. 31st IEEE/ACM Int. Conf. Automated Software Engineering, 2016, pp. 75–86

[116] Peters, F., Menzies, T., Layman, L.: 'Lace2: better privacy-preserving data sharing for cross project defect prediction'. Proc. 37th Int. Conf. Software Engineering, 2015, pp. 801–811

[117] Mende, T., Koschke, R.: 'Effort-aware defect prediction models'. Proc. 14th European Conf. Software Maintenance and Reengineering, 2010, pp. 107–116

[118] Kamei, Y., Matsumoto, S., Monden, A., *et al.*: 'Revisiting common bug prediction findings using effort-aware models'. Proc. IEEE Int. Conf. Software Maintenance, 2010, pp. 1–10

[119] Zhou, Y., Xu, B., Leung, H., *et al.*: 'An in-depth study of the potentially confounding effect of class size in fault prediction', *ACM Trans. Softw. Eng. Methodol.*, 2014, **23**, (1), p. 10

[120] Yang, Y., Zhou, Y., Lu, H., *et al.*: 'Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study', *IEEE Trans. Softw. Eng.*, 2015, **41**, (4), pp. 331–357

[121] Sarkar, S., Kak, A.C., Rama, G.M.: 'Metrics for measuring the quality of modularization of large-scale object-oriented software', *IEEE Trans. Softw. Eng.*, 2008, **34**, (5), pp. 700–720

[122] Zhao, Y., Yang, Y., Lu, H., *et al.*: 'An empirical analysis of package-modularization metrics: implications for software fault-proneness', *Inf. Softw. Technol.*, 2015, **57**, (1), pp. 186–203

[123] Yang, Y., Zhou, Y., Liu, J., *et al.*: 'Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models'. Proc. 24th ACM SIGSOFT Int. Symp. Foundations of Software Engineering, 2016, pp. 157–168

[124] Yang, Y., Harman, M., Krinke, J., *et al.*: 'An empirical study on dependence clusters for effort-aware fault-proneness prediction'. Proc. 31st IEEE/ACM Int. Conf. Automated Software Engineering, 2016, pp. 296–307

[125] Ma, W., Chen, L., Yang, Y., *et al.*: 'Empirical analysis of network measures for effort-aware fault-proneness prediction', *Inf. Softw. Technol.*, 2016, **69**, pp. 50–70

[126] Zhao, Y., Yang, Y., Lu, H., *et al.*: 'Understanding the value of considering client usage context in package cohesion for fault-proneness prediction', *Autom. Softw. Eng.*, 2017, **24**, (2), pp. 393–453

[127] Bennin, K.E., Keung, J., Monden, A., *et al.*: 'Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models'. Proc. IEEE 40th Annual Computer Software and Applications Conf., 2016, pp. 154–163

[128] Panichella, A., Alexandru, C.V., Panichella, S., *et al.*: 'A search-based training algorithm for cost-aware defect prediction'. Proc. 2016 on Genetic and Evolutionary Computation Conf., 2016, pp. 1077–1084

[129] Shepperd, M., Bowes, D., Hall, T.: 'Researcher bias: the use of machine learning in software defect prediction', *IEEE Trans. Softw. Eng.*, 2014, **40**, (6), pp. 603–616

[130] Tantithamthavorn, C., McIntosh, S., Hassan, A.E., *et al.*: 'Comments on 'researcher bias: the use of machine learning in software defect prediction'', *IEEE Trans. Softw. Eng.*, 2016, **42**, (11), pp. 1092–1094

[131] Ghotra, B., McIntosh, S., Hassan, A.E.: 'Revisiting the impact of classification techniques on the performance of defect prediction models'. Proc. 37th Int. Conf. Software Engineering, 2015, pp. 789–800

[132] Tantithamthavorn, C., McIntosh, S., Hassan, A.E., *et al.*: 'Automated parameter optimization of classification techniques for defect prediction models'. Proc. 38th Int. Conf. Software Engineering, 2016, pp. 321–332

[133] Bowes, D., Hall, T., Petrić, J.: 'Software defect prediction: do different classifiers find the same defects?' *Softw. Qual. J.*, 2017, Available at https://doi.org/10.1007/s11219-016-9353-3

[134] Bowes, D., Hall, T., Gray, D.: 'Dconfusion: a technique to allow cross study performance evaluation of fault prediction studies', *Autom. Softw. Eng.*, 2014, **21**, (2), pp. 287–313

[135] Malhotra, R., Khanna, M.: 'An exploratory study for software change prediction in object-oriented systems using hybridized techniques', *Autom. Softw. Eng.*, 2017, **24**, (3), pp. 673–717

[136] Zhang, F., Hassan, A.E., McIntosh, S., *et al.*: 'The use of summation to aggregate software metrics hinders the performance of defect prediction models', *IEEE Trans. Softw. Eng.*, 2017, **43**, (5), pp. 476–491

[137] He, P., Li, B., Liu, X., *et al.*: 'An empirical study on software defect prediction with a simplified metric set', *Inf. Softw. Technol.*, 2015, **59**, pp. 170–190

[138] Yang, Y., Zhao, Y., Liu, C., *et al.*: 'An empirical investigation into the effect of slice types on slice-based cohesion metrics', *Inf. Softw. Technol.*, 2016, **75**, pp. 90–104

[139] Jaafar, F., Guéhéneuc, Y.G., Hamel, S., *et al.*: 'Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults', *Empir. Softw. Eng.*, 2016, **21**, (3), pp. 896–931

[140] Chen, L., Ma, W., Zhou, Y., *et al.*: 'Empirical analysis of network measures for predicting high severity software faults', *Sci. China Inf. Sci.*, 2016, **59**, (12), pp. 1–18

[141] Kamei, Y., Fukushima, T., McIntosh, S., *et al.*: 'Studying just-in-time defect prediction using cross-project models', *Empir. Softw. Eng.*, 2016, **21**, (5), pp. 2072–2106

[142] Petrić, J., Bowes, D., Hall, T., *et al.*: 'Building an ensemble for software defect prediction based on diversity selection'. Proc. 10th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement, 2016, p. 46

[143] Liu, W., Liu, S., Gu, Q., *et al.*: 'Empirical studies of a two-stage data preprocessing approach for software fault prediction', *IEEE Trans. Reliab.*, 2016, **65**, (1), pp. 38–53

[144] Xu, Z., Liu, J., Yang, Z., *et al.*: 'The impact of feature selection on defect prediction performance: an empirical comparison'. Proc. IEEE 27th Int. Symp. Software Reliability Engineering, 2016, pp. 309–320

[145] Mende, T.: 'Replication of defect prediction studies: problems, pitfalls and recommendations'. Proc. 6th Int. Conf. Predictive Models in Software Engineering, 2010, pp. 1–10