The Seybold Report

ISSN 1533-9211

DOI 10.5281/zenodo.7318104

SOFTWARE DEFECT PREDICTION TECHNIQUES IN SOFTWARE ENGINEERING: A REVIEW

ABDUALMAJED A. G. AL-KHULAIDI[1], AHMED A. ABDU[2], ADEL A. NASSER[3, 4*], HAKIM A. ABDU[5], MIJAHED ALJOBER[4] and MUNEER A. S. HAZZA[5]

[1] Department of Computer Science, Faculty of Computer and Information Technology, Sana'a University, Sana'a, Yemen.
[2] School of Software, Northwestern Polytechnical University, Xian, China.
[3]Department of Information Systems and Computer Science, Faculty of Sciences, Sa'adah University, Sa'adah, Yemen.
[4]Modern Specialized College of Medical and Technical Sciences, Sana'a, Yemen.
[5]School of Computer Science & Information Technology, Dr. Babasaheb Ambedkar Marathwada University, India.
[6]Faculty of Computer and Information Systems, Thamar University, Thamar, Yemen.
[*]Corresponding author: adel@saada-uni.edu.ye

Abstract

Defect prediction is one of the significant challenges in the software development lifecycle for improving software quality and reducing program testing time and cost. Developing a defect prediction model is a difficult task, and several techniques have been developed over time. Previous reviews focused on defect prediction in general, and none have specifically addressed defect prediction based on the semantic representation of programs from source code. This review presents a comprehensive and holistic survey of software defect research over three decades, covering motivations, datasets, state-of-the-art techniques, challenges, and future research directions. We specifically concentrate on source code semantic-based methods. We also give particular attention to the techniques based on semantic features because it presents the field's current state of the art. We focus on the process of cross-project defect prediction (CPDP), within-project defect prediction (WPDP), and the most recently used datasets. Defect datasets for 60 projects in different programming languages (C, Java, and C++) are presented and analyzed. Open issues are studied, and potential research directions in defect prediction are proposed to supply the reader with a point of reference for important topics that deserve study.

Keywords: Cross project, deep learning, Semantic features, Software defect prediction, within project

1) INTRODUCTION

Software Defect Prediction (SDP) is one of the most critical study fields in software engineering. Software defect prediction models supply a list of fault-prone software products, enabling the quality assurance team to spend limited investigating resources and testing software products efficiently (Song Wang, et al., 2016). They can influence more time-consuming and expensive quality assurance approaches (e.g., human inspection). Boehm and Basili (Boehm & Basili, 2007) report that previous research has found that most software defects occur in a small number of software modules. As a result, many previous studies have been devoted to determining how to prioritize software quality efforts.

As the scale and complexity of software projects grow, SDP mechanisms are becoming increasingly important. SDP models help developers and testers identify whether a software

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

system is defective early in the project life cycle, allowing them to effectively manage test resources, optimize the testing process, and increase software quality (Wan, et al., 2018). Earlier on, researchers used metric-based techniques to solve the problem, relying on different data types, such as previous defects, code metrics (lines of code, complexity), or process metrics (recent activity, number of changes). Subramanyam et al. (Subramanyam & Krishnan, 2003) and Nagappan et al. (Nagappan, Ball, & Zeller, 2006) presented empirical evidence for using OO design complexity measures in determining software defects, specifically a subset of the CK suite. Moser et al. (Moser, Pedrycz, & Succi, 2008) evaluated the efficiency of static code and changed attributes in defect prediction. Hassan (Hassan, 2009) created a defect prediction model using complexity measures based on the source code change process rather than the source code itself. In some cases, the researchers' conclusions were inconsistent; for example, in opposition to the decision of Fenton et al. (Norman E. Fenton & Ohlsson, 2000), Gyimóthy et al. (Gyimóthy, Ferenc, & Siket, 2005) showed positive outcomes in the case of size metrics.

As an alternative to standard metric-based methodologies, semantic strategies have evolved for defect prediction in recent years. This method predicts defects directly from the project's source code rather than relying on metric generator analysis (Liang, Yu, Jiang, & Xie, 2019). These approaches involve employing unsupervised learning for fitting an obstetric model to create features that perform well for reconfiguring the source code as the sequential structure or abstract syntax trees in the input data set. Other classification algorithms will use these features similarly to metric-based defect prediction, but with metrics replaced by the features of unsupervised learning.

Several previous reviews and surveys of the field have been performed; Fenton et al. (Norman E Fenton & Neil, 1999) Fenton provided a critical study of defect prediction models, which have shown that many theoretical mistakes and methodological have been made. Catal and Diri (Catal & Diri, 2009) reviewed studies published before 2009 concerning datasets, metrics, and methods. Hall et al. (Hall, Beecham, Bowes, Gray, & Counsell, 2011) analyzed the articles on fault prediction published between 2000 and 2010 using a systematic literature review. Wahono (Wahono, 2015) identified and analyzed defect prediction research trends, frameworks, methodologies, and datasets from 2000 to 2013. Rathore and Kumar (Rathore & Kumar, 2019) searched through many online libraries and identified the relevant studies published between 1993 and 2017. This review differs from others in the following ways:

- Semantics features analysis: This review is characterized by a review of previous studies based on the semantics features of source code, while previous reviews did not allocate space to these studies
- Timeframes: This review is more holistic and contemporary because it covers research published between 1970 and 2021.
- Comprehensiveness: Unlike most previous reviews, our review does not depend entirely on search engines; we read the articles from related journals and conferences one by one. As a result, we studied a wide range of papers.

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

- In this review, we make four significant contributions:
- This review provides an updated survey of software defect prediction techniques: history, present, and future challenges. The approaches reviewed were analyzed and summarized under various conditions.
- This review presents and analyze the most used defect datasets for 60 projects in different programming languages (C, Java, C++) and divide them under their metrics and defect labels.
- This review has summarized more than 115 scientific publications on software defect prediction and its solemn data gathering and preprocessing problems from 1970 to 2021. These publications have been classified according to various approaches: simple metrics, Just-In-Time (JIT), cross-project prediction, semantic techniques with deep learning, and hybrid models. We give particular focus approaches based on semantic features, which are currently employed for predicting software defects.
- This review highlights some new trends and future challenges for software defect prediction technology by paying particular attention to the aspect of source code semantic-based techniques.

This review is structured as follows. In Section 2, we introduce the related work. The process of SDP is explained in Section 3. Section 4 describes the available datasets and evaluation measures. Section 5 presents software defect prediction based on semantic features. In Section 6, we discuss the SDP challenges and future directions. Finally, in Section 7, we conclude our review.

2) RELATED WORK

Before going on, we first discuss the similar reviews and surveys on SDP. Fenton et al. (Norman E Fenton & Neil, 1999) provided a review of the state-of-the-art. They also advocated for 'software decomposition' research to evaluate fault introduction hypotheses and develop a better software engineering science. Catal and Diri (Catal & Diri, 2009) reviewed 74 SDP publications and compared studies released before and after 2005 in performance metrics, techniques, and datasets. Hall et al. (Hall, et al., 2011) identified a review for defect prediction articles published between 2000 and 2010. The criteria they set and applied summarized the qualitative and quantitative results of 36 research that provided sufficient contextual information. They also investigated how the independent variables, the context of models, and the modeling approaches used affect the defect prediction model performance.

Hosseini et al. (Hosseini, Turhan, & Gunarathna, 2017) summarized and synthesized existing CPDP studies to identify the type of performance evaluation criteria, modeling techniques, approaches, and independent variables employed in CPDP model development. They conducted a comprehensive literature review with meta-analysis to achieve their study goal and answer the research questions. Moreover, their study aimed to compare the achievement of WPDP models with that of CPDP models. Malhotra (Malhotra, 2015) reviewed works in the

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

literature that used machine learning approaches for SDP from January 1991 to October 2013. They evaluated the effectiveness of machine learning algorithms for SDP in existing research and identified seven machine-learning technique categories. Their results showed the machine learning ability to predict whether a module or class is defect-prone. Li et al. (Z. Li, Jing, & Zhu, 2018) analyzed and discussed almost 70 relevant defect prediction publications from January 2014 to April 2017. They summarized the selected papers into four aspects: effort-aware prediction, data manipulation, machine learning algorithms, and empirical studies. Li et al. (N. Li, Shepperd, & Guo, 2020) carried out a systematic review identifying and analyzing the studies published between 2000 and 2018. The confusion matrices were recalculated, and they used Matthews Correlation Coefficient (MCC) as their primary performance indicator. As a result, their meta-analysis has shown that supervised and unsupervised models are comparable for both CPDP and WPDP. Rathore and Kumar (Rathore & Kumar, 2019) searched many digital libraries to locate the relevant publications published between 1993 and 2017. Akimova et al. (Akimova, et al., 2021) presented the software defect prediction based on deep learning approaches, such as the LSTM architecture. They identified the primary challenges of defect prediction as a lack of data and a complex context, and they proposed solutions to these issues. These publications are categorized into defect prediction methodologies, software metrics, and data quality concerns. Taxonomic classifications of various methods, as well as their observations, have been presented for each class.

This review presents the history of defect prediction technology, the current approaches, and future challenges. Given the relatively short history of work in SDP based on the source code semantic information of programs, the first paper published on this point was in 2016 by Wang et al. (Song Wang, et al., 2016). As a result, many publications based on source code semantic information were not included in previous reviews. Given the practical implications abovementioned, this powerfully demonstrates a research gap, justifying our motivation to complete this review.

3) SOFTWARE DEFECT PREDICTION PROCESS

3.1 Within-project defect prediction

In the case of the Within-Project Defect Prediction, researchers train SDP models using historical data from the same project and then use them to predict faults in future releases. On the other hand, WPDP models are built and tested in the same project (Nam, 2014). The first stage in building a WPDP model is creating instances from historical project data (archive of programs) such as version control systems, e-mail archives, issue tracking systems, etc. Based on prediction granularity, instances can represent a software system, a code file, package, function, class, and a source code modification. The instances have some features (attributes) extracted from historical data labeled as defective or clean. As shown in Figure 1, instances generated from the historical data are marked with ‘1’ (defective), ‘0’ (clean). After creating instances with attributes and labels, preprocessing techniques are applied to divide the datasets into training sets and test sets. Finally, the prediction model trains on the final training set, as shown in Figure 1. The prediction model predicts if a new instance is defective or not.

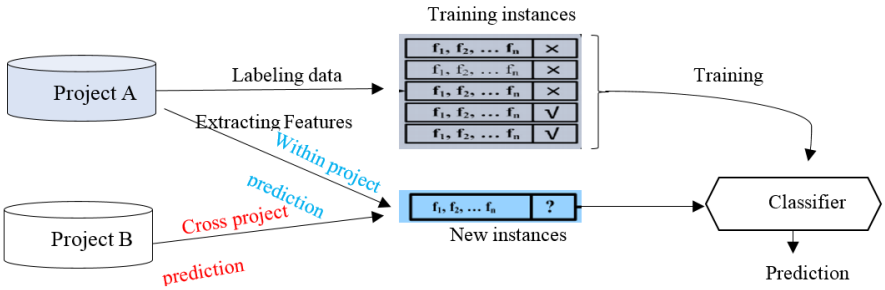
The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

Here we will summarize some significant research works. Elish et al. (Elish & Elish, 2008) used an SVM to predict defect-prone and compared its prediction accuracy with eight machine learning algorithms on the NASA dataset. Lu et al. (Lu, Kocaguneli, & Cukic, 2014) use active learning to estimate defects; they also utilize feature compression approaches to reduce the features in defect data. Li et al. (M. Li, Zhang, Wu, & Zhou, 2012) proposed ACoForest, a semi-supervised technique that can sample the most helpful prediction modules for learning. Seiffert et al. (Seiffert, Khoshgoftaar, Van Hulse, & Folleco, 2014) evaluated 11 methods and seven data sampling approaches and discovered that data noise and class imbalance negatively impact the prediction accuracy. Jin.(C Jin, 2011) presented how to predict software fault-proneness using a Quantum Particle Swarm Optimization and ANN. ANN is used to categorize software modules into defect-prone or non-defect-prone groups, while the QPSO reduces dimensionality.

3.2 Cross-project defect prediction

Cross-project defect prediction is a practical method to predict defects in projects with limited historical data or new projects (Gong, Jiang, Bo, Jiang, & Qian, 2019). CPDP model is first trained using existing training data from one project with a sufficient dataset and then used to predict the defects of a new project with a limited dataset (Cong Jin, 2021a).

Figure 1 : Process of software defect prediction (Song Wang, Liu, & Tan, 2016)



The main challenge in CPDF is transferring efficient metrics from a source project. And using these features to create the prediction model in a target project. These problems are because the development languages, application scenarios, and the developers' skills of the source and target projects are often quite different (Zhu, Zhang, Ying, & Wang, 2020).

Despite the distinctions mentioned above between various software systems, there are some similarities, such as similar coding styles, the same development language, architecture similarity, etc. These similarities indicate the connection between source and target projects. Figure 1 shows the main processes of CPDP model and how the dataset is generated and extracted. Many CPDP models have recently been presented, and many academics have paid attention to them; some significant research efforts will be listed below. Jin (Cong Jin, 2021a) used kernel-twin support vector machines for performing domain adaptation to meet various training data distributions. In this study, the author used KTSVMs with domain adaptation functions as the CPDP model. Ryu et al. (Ryu, Choi, & Baik, 2016) studied whether class imbalance training can assist with CPDP models. The similarity weights and cost of

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

asymmetric misclassification generated from distribution features are strongly linked in their approach to determining the optimal resampling strategy. Zhang et al. (F. Zhang, Zheng, Zou, & Hassan, 2016) used three public datasets (AEEEM, NASA, and PROMISE) from 26 projects to compare the achievement of supervised and unsupervised classifiers. Among five supervised classifiers (decision tree, random forest, logistic regression, naive Bayes, and logistic tree model) and five unsupervised classifiers (fuzzy C-means, partition medoids, k-means, spectral clustering, and neural gas), their proposed connectivity-based classifier is regarded as one of the best.

Zimmermann et al. (Zimmermann, Nagappan, Gall, Giger, & Murphy, 2009) investigated CPDP models on a large scale. Their research employed cross-project to provide 622 predictions on 12 real-world systems. Their findings suggested that CPDP is a challenging task and that relying on models from similar projects in the same field or studies utilizing the same approach would not produce reliable results. Turhan et al. (Turhan, Menzies, Bener, & Di Stefano, 2009) also investigated CPDP. They get results similar to Zimmermann et al. (Zimmermann, et al., 2009) in a first experiment, a defect prediction model trained from one project performs poorly on future projects, especially in terms of the false positives rate. After a deeper investigation, they discovered that the bad results were due to irrelevant details; for example, the model learned much irrelevant information from previous projects. As a result, Turhan et al. (Turhan, et al., 2009) used a relevancy filtering strategy, which significantly reduced the number of false positives. The feasibility of CPDP is another exciting issue. According to most reviewed research, CPDP is challenging to produce, and only a few CPDP combinations work. There have been several studies on the feasibility of CPDP; He et al. (Z. He, Shu, Yang, Li, & Wang, 2012) studied the feasibility of CPDP, concentrating on training data selection and defect prediction in a cross-project scenario. Table 1 shows the list of studies was made on within/cross projects.

Table 1: List of studies was made on within/cross projects

Studies	Project Type		Datasets	Techniques
	Within	Cross		
Li et al. (J. Li, He, Zhu, & Lyu, 2017)	√		PROMISE	Convolutional Neural Network
Elish et al. (Elish & Elish, 2008)	√		NASA (CM1, PC1, KC1, and KC3)	Support vector machine
Li et al. (M. Li, et al., 2012)	√		PROMISE	Semi-supervised method
Zhou et al. (Zhou, Sun, Xia, Li, & Chen, 2019)	√		NASA, PROMISE, AEEEM, and Relink	Deep forest model
Pan et al. (Pan, Lu, Xu, & Gao, 2019)	√		PROMISE	Convolutional neural network
Jin. (Cong Jin, 2021a)		√	AEEEM and Kamei	Kernel twin support vector machines (KTSVMs)
Ryu et al. (Ryu, et al., 2016)		√	NASA and SOFTLAB	Support vector machine
Xu et al. (Z. Xu, et al., 2019)		√	NASA and SOFTLAB	Transfer learning method
Chen et al. (J. Chen, Hu, Yang, Liu, & Xuan, 2020)		√	ReLink, AEEEM, SOFTLAB, and MORPH	Collective transfer learning
Zhang et al. (F. Zhang, Zheng, et al., 2016)		√	AEEEM, NASA, and PROMISE	Connectivity-based classifier (via spectral clustering)
He et al. (P. He, Li, Zhang, & Ma, 2014)		√	PROMISE	Logistic Regression and Naive Bayes
Ni et al. (Ni, et al., 2017)		√	ReLink and AEEEM	Hybrid-Data Clusters by feature selection
Canfora et al. (Canfora, et al., 2013)		√	Promise	Logistic regression and Genetic algorithm.
Ma et al. (Ma, Luo, Zeng, & Chen, 2012)		√	NASA and SOFTLAB	Transfer Naive Bayes
Liu et al. (Liu, Yang, Xia, Yan, & Zhang, 2019)		√	PROMISE	Transfer learning method
He et al. (P. He, Li, & Ma, 2014)		√	PROMISE, ReLink and AEEEM	A simple approach based on object class mapping.
Xia et al. (Xia, Lo, Pan, Nagappan, & Wang, 2016)		√	PROMISE	Genetic learning and Ensemble algorithm
Yang et al. (Xingguang Yang, Yu, Fan, Shi, & Chen, 2019)		√	Kamei	k-medoids and logistic regression
Czibula et al. (Czibula, Marian, & Czibula, 2014)	√	√	NASA	Collective transfer learning
Zhu et al. (Zhu, et al., 2020)	√	√	Kamei	Naive Bayes algorithm
Wei et al. (Wei, Hu, Chen, Xue, & Zhang, 2019)	√	√	NASA	Support vector machine
Gong et al. (Gong, et al., 2019)	√	√	PROMISE and NASA	Stratification embedded in nearest neighbor (STr-NN)
Zhu et al. (Zhu, Ying, Zhang, & Zhu, 2021)	√	√	PROMISE, ReLink, and AEEEM	CNN and kernel extreme learning machine
Jin et al. (Cong Jin & Jin, 2015)	√	√	NASA	Hybrid ANN and Quantum Particle Swarm Optimization.
Tran et al. (Tran, Hanh, & Binh, 2019)	√	√	NASA	Random decision forests
Wu et al. (F. Wu, et al., 2018)	√	√	MORPH, NASA, ReLink, and AEEEM.	Cost-sensitive kernelized semi-supervised dictionary learning
Jin et al. (Cong Jin, 2021b)	√	√	NASA	Distance metric learning
Jiang et al. (Jiang, Cukic, & Menzies, 2008)	√	√	NASA	Naïve Bayes
Felix et al. (Felix & Lee, 2017)	√	√	PROMISE	Regression model
Seliya et al. (Seliya, Khoshgoftaar, & Zhong, 2005)	√	√	NASA	Semi-supervised clustering
Jing et al. (X.-Y. Jing, Wu, Dong, & Xu, 2016)	√	√	AEEEM	Subclass discriminant analysis and semi supervised transfer

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

4) AVAILABLE DATASETS AND EVALUATION MEASURES

4.1 Available datasets

Researchers created several public defect datasets and used them to train defect prediction models. Based on these datasets, defect prediction models can predict either defect-proneness (classification) or the defect count (regression). In this review, we collected the most used defect datasets. Table 2 summarizes the commonly used datasets.

Table 1 : Summary of datasets

Dataset	Avg. instances	Avg. defect-prone%	Remarks
AEEEM	1074	19.40	The AEEEM dataset was created by D'Ambros et al. (D'Ambros, Lanza, & Robbes, 2012) to implement class-level defect prediction.
NASA MDP	1585	17.67	NASA MDP came from National Aeronautics and Space Administration (NASA) mission-critical software projects. There are officially 13 data sets in the NASA MDP repository.
Kamei	37902	21.11	Kamei dataset published by Kamei et al. (Kamei, et al., 2012), Kamei dataset was created from 6 open-source software (i.e., Columba, Bugzilla, Mozilla, JDT, Eclipse, and PostgreSQL), and its goal is to develop a risk model for modification.
ReLink	217	37.79	Wu et al. (R. Wu, Zhang, Kim, & Cheung, 2011) created ReLink dataset from three projects: Safe, Apache, ZXing, each with 26 metrics.
SOFTLAB	86	15.20	SOFTLAB came from a Turkish technology company that specialized in domestic appliance applications. SOFTLAB contains five projects (ar1, ar3, ar4, ar5, ar6), including 29 static code features.
JIRA	1483	14.01	Yatish et al. (Yatish, Jiarpakdee, Thongtanunam, & Tantithamthavorn, 2019) have created a new repository for defect datasets called JIRA. JIRA dataset is a collection of models from 9 open-source systems.
PROMISE	499	18.52	The PROMISE dataset was created by Jureczko (Jureczko & Madeyski, 2010), which contains the characteristics of most different projects.
MORPH	338	24	The MORPH group was presented by Peters et al. (Peters & Menzies, 2012), which contains defect data sets from various open-source projects utilized in a study on the issue of dataset privacy in defect prediction.

4.2 Evaluation measures

The prediction model has four possibilities for predicting the result of a code change: 1) predict a defective code change as a defect (True positive, TP); 2) predict a defective code change as no defect (False Positive, FP); 3) Predict a nondetective code change as no defect (True Negative, TN); 4) Predict a nondetective code change as defective (False Negative, FN) (Abdu, et al., 2022). According to these four possibilities, the prediction model results in the test set and can calculate the performance indicators such as recall, F-measure, precision, G-measure, accuracy, balance, and G-mean.

Precision refers to the proportion of all correctly classified defective changes to all classified as faulty changes; as introduced by Menzies et al. (Menzies, Dekhtyar, Distefano, & Greenwald, 2007), precision is a volatile performance measure when datasets have a low ratio of defects.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

Recall: refers to the proportion of all correctly classified defective changes to all truly faulty changes. This indicator is significant for defect prediction, as prediction models aim to detect as many false instances as possible.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F-measure: It is a comprehensive performance indicator that combines the precision rate and the recall rate. It is the harmonic average of the precision rate and the recall rate:

$$\text{F} - \text{measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

G-measure: According to Peters et al. (Peters, Menzies, Gong, & Zhang, 2013), G-measure is the harmonic mean of recall and 1-PF, which is calculated as:

$$\text{G} - \text{measure} = \frac{2 \times \text{recall} \times (1 - \text{pf})}{\text{recall} + (1 - \text{pf})}$$

PF: represents specificity in Jiang et al. (Jiang, Cukic, & Ma, 2008), which is defined as:

$$\text{Pf} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Accuracy: Refers to the ratio of correctly classified code changes to all code changes:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

G-mean: Geometric mean is used to measure the overall predictors in the imbalanced context. Wang and Yao (Shuo Wang & Yao, 2013). It calculates the G-mean of recall and 1−Pf. In the SDP case, G-mean is defined as:

$$\text{G} - \text{mean} = \sqrt{\text{recall} \times (1 - \text{pf})}$$

Balance: It determines the Euclidean distance between Pf = 0 and recall= 1 on the ROC sweet spot. As a result, the higher balances approach the ideal sweet spot of Pf = 0 and recall= 1 more closely. Balance is defined as:

$$\text{Balance} = 1 - \frac{\sqrt{(0 - \text{Pf})^2 + (1 - \text{recall})^2}}{\sqrt{2}}$$

AUC: Area Under Curve refers to the area under the ROC curve and is a commonly used evaluation indicator in real-time defect prediction research (Lessmann, Baesens, Mues, & Pietsch, 2008). In practical applications, developers will use the output results of the prediction model to schedule work, and AUC is suitable for evaluating this scheduling.

MCC: Matthews correlation coefficient is defined as the measure of true and false positives and negatives (F. Zhang, Mockus, Keivanloo, & Zou, 2016). With values in [-1, 1], MCC estimates the relation between measured and predicted binary classifications (X. Jing, Wu,

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

Dong, Qi, & Xu, 2015). MCC is defined as follows:

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

Table 3: Software metrics and performance measures across the categorical studies

Studies	Type of metrics	Performance measures
Mizuno et al. (Mizuno & Hirata, 2014)	Source code text	Precision, Recall, F-measure, Accuracy, and G-mean
Kamei et al. (Kamei, et al., 2016)	Process	Precision, Recall, F-measure, and AUC
Herbold (Herbold, 2013)	OO+SCM+LOC	Precision and Recall
Nam et al. (Nam, Pan, & Kim, 2013)	OO+SCM Process+LOC	F-measure
Dalla Palma (Dalla Palma, Di Nucci, Palomba, & Tamburri, 2021)	Product and Process	AUC and MCC
Cruz et al. (Cruz & Ochimizu, 2009)	OO	G-mean
Canfora et al. (Canfora, et al., 2015)	OO+LOC	Precision, Recall, and AUC
Majd et al. (Majd, Vahidi-Asl, Khalilian, Poorsarvi-Tehrani, & Haghighi, 2020)	Multiple Combinations	Precision, Recall, F-measure, and Accuracy
Turhan et al. (Turhan, Mısırlı, & Bener, 2013)	OO+SCM +LOC	Recall, PF, and Balance
Zhang et al. (F. Zhang, Mockus, et al., 2016)	OO+SCM +LOC	Precision, G-measure, PF, AUC, and MCC
Panichella et al. (Panichella, Oliveto, & De Lucia, 2014)	OO+LOC	AUC
He et al. (P. He, Li, Liu, Chen, & Ma, 2015)	OO+SCM +LOC	Precision, Recall, and F-measure
Zhu et al. (Zhu, et al., 2021)	Multiple Combinations	F-measure, G-measure, AUC, and MCC
Jing et al. (X. Jing, et al., 2015)	OO+SCM Process+LOC	Recall, PF, and MCC
He et al. (Z. He, et al., 2012)	OO+SCM +LOC	Precision, Recall, and F-measure
Nam et al. (Nam, Fu, Kim, Menzies, & Tan, 2017)	OO+SCM Process+LOC	AUC
Uchigaki et al. (Uchigaki, Uchida, Toda, & Monden, 2012)	SCM+LOC	AUC
Jin (Cong Jin, 2021a)	Multiple Combinations	Precision, Recall, and F-measure
Peters et al. (Peters, Menzies, & Layman, 2015)	OO+SCM +LOC	Recall, F-measure, PF, and G-mean
Turhan et al. (Turhan, et al., 2009)	SCM+LOC	Recall, PF, and Balance
Ryu et al. (Ryu, et al., 2016)	SCM+LOC	G-mean and AUC
Chen et al. (L. Chen, Fang, Shang, & Tang, 2015)	OO+SCM +LOC	Recall, G-measure, PF, and MCC
Zhao et al. (Zhao, Shang, Zhao, Zhang, & Tang, 2019)	Multiple Combinations	PF, Accuracy, and MCC
Zhang et al. (Y. Zhang, Lo, Xia, & Sun, 2015)	OO+SCM +LOC	F-measure
Ryu et al. (Ryu, Jang, & Baik, 2015)	code+LOC	Recall, PF, and Balance
Limsettho et al. (Limsettho, Bennin, Keung, Hata, & Matsumoto, 2018)	Multiple Combinations	F-measure, G-measure, and Balance
Rahman et al. (Rahman, Posnett, & Devanbu, 2012)	Multiple Combinations	Precision, Recall, F-measure, and Accuracy
Laradji et al. (Laradji, Alshayeb, & Ghouti, 2015)	Multiple Combinations	G-mean and AUC

According to the analysis of the previous studies, we can say that Precision, Recall, and F-measure are the metrics that have been widely adopted to evaluate defect prediction techniques.

5) SOFTWARE DEFECT PREDICTION BASED ON SEMANTIC FEATURES

As an alternative to standard metric-based methodologies, semantic strategies have evolved for defect prediction in recent years. In 2016, Wang et al. (Song Wang, et al., 2016) supposed a new approach to predict defects directly from the project's source code rather than relying on metric generator analysis. These approaches involve employing unsupervised learning for fitting an obstetric model to create features that perform well for reconfiguring the source code as the sequential structure or abstract syntax trees in the input data set. Other classification algorithms will use these features similarly to metric-based defect prediction but with metrics replaced by the features of unsupervised learning (Abdu, et al., 2022).

5.1 Code representation

To work with the problem of SDP, researchers require a representation of the source code. Furthermore, because many machine learning approaches work with vectors, the code representation should be as simple as a vector. Moreover, this representation should include all relevant information (Song Wang, Liu, Nam, & Tan, 2018). The source code can be represented in different ways. Moreover, different granularities are required for different purposes; for example, token-level embedding is needed for code completion, but function clone detection requires function embedding. Different levels of granularity are used to solve SDP problems,

ISSN 1533-9211

DOI 10.5281/zenodo.7318104

such as components, methods, classes, subsystems, and changes (Allamanis, Barr, Devanbu, & Sutton, 2018).

One approach is to generate the vectors from the code handcrafted features. This method proposes that an expert generates a set of attributes and then determines the best among them (Dam, et al., 2018). Statistical code metrics are usually included in these features, such as code complexity, code size, process metrics, or code churn. The three types of code representation now in use are token-based representations, graph-based representations, and AST-based representations (Hua, Sui, Wan, Liu, & Xu, 2020).

In the process of code representation based on a token, the code fragment is divided into tokens, and the bag of words (BOW) term is used to count the frequency of each token that appears in a document (Sajnani, Saini, Svajlenko, Roy, & Lopes, 2016). Since a code line generally includes several tokens, token-based representation demands more CPU time and memory than line-by-line comparison. However, token-based representation can utilize various transformations, by which many code segments are identified as clone pairs, and differences in coding style are efficiently avoided (Kamiya, Kusumoto, & Inoue, 2002).

AST is a type of tree that is used to represent the source code's abstract syntactic structure. It has been used by software engineering tools and programming languages (Baxter, Yahin, Moura, Sant'Anna, & Bier, 1998). Compared with the simple source code, AST is conceptual and does not contain delimiters and punctuation. On the other hand, AST can describe the source code's syntactic structure and linguistic information (J. Zhang, et al., 2019).

In the state of code representation based on the graph control flow CFGs (Allen, 1970), such as program dependence graphs (Gabel, Jiang, & Su, 2008), graph representation of programs (Yousefi, Sedaghat, & Rezaee, 2015), and call graphs, can be used to display the structural information of the code. Each code representation technique is well designed to gather specific data that can be used alone or with other methods to produce a more extensive result. After completing the code representation, metrics carrying important information from the source code will be gathered, which can be used in code clone detection, defect prediction, automatic program repair, etc. (H. Wang, Zhuang, & Zhang, 2021).

5.2 Deep learning model for software defect prediction based on semantic features

Deep learning models is currently gaining popularity in the software defect prediction based on analysis source code semantic features. Figure 2 shows the deep learning model process for the software defect prediction based on semantic features, which displays the following main steps.

- Extracting features from source code repositories to compose the defect datasets consisting of software metrics.
- Performing the preprocessing on the software metrics and then acquiring the training data and test data.
- The third step uses deep learning techniques such as DBN, LSTM, CNN, Transformer,

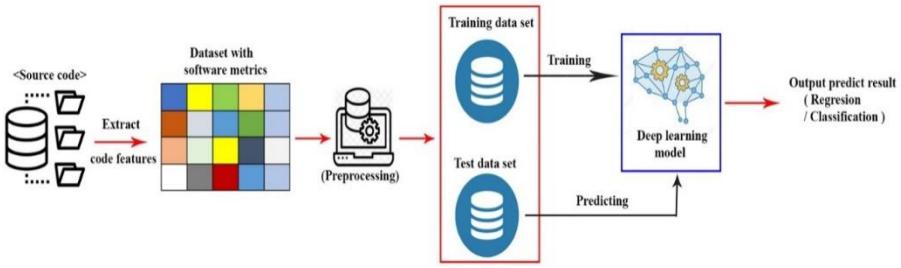
ISSN 1533-9211

DOI 10.5281/zenodo.7318104

and other deep learning algorithms to build an SDP model.

- The fourth step is to train a model using the training dataset; after the model is trained, use the test dataset to test the prediction results and do some predicting cases, i.e., predict whether a code module is defective or clean.
- The last step is to evaluate the model's performance using evaluation measures, such as balance, recall, precision, accuracy, and other performance measures.

Figure 2: Software defect prediction based on deep learning



Deep Belief Network is a type of neural network that is represented as a generative graphical model, built up of multiple layers of implicit variables (hidden units) with links between layers but not between the units inside each layer (Hinton, Osindero, & Teh, 2006). Several works use DBN approach in defect prediction, Yang et al. (Xinli Yang, Lo, Xia, Zhang, & Sun, 2015) suggested a deep learning model for JIT defect prediction. Their approach uses a DBN to extract a collection of expressive attributes from an input set of basic change measures. It then uses logistic regression to train a classifier relying on the retrieved features. Wang et al. (Song Wang, et al., 2018) use the DBN to predict defects based on the source code semantics information; the authors created a DBN to learn semantic information from the source code. The AST and code changes of the programs are used as input for the network in the situations of change-level and file-level prediction, respectively.

Long short-term memory is a recurrent neural network, and it can process entire data sequences and single data points (Hochreiter & Schmidhuber, 1997). LSTM unit is made of input gate, output gate, cells, and forget gate. The three gates control the information flow into and out of the cell, and the cell recalls values across arbitrary time intervals. Dam et al. (Dam, et al., 2019) propose a new deep learning tree-based defect prediction model. Their model is based on a tree-structured LSTM network that perfectly matches the AST source code representation. As the code representation for statement-level SDP, Majd et al. (Majd, et al., 2020) merged the token sequence of each line of code with the static metric. Their study employed LSTM to encode these sequences, and the model outperformed the RF classifier. Deng et al. (Deng, Lu, & Qiu, 2020) used LSTM to learn the semantic metrics from the source code. They go through each file's AST and feed them into an LSTM network to automatically extract the program's semantic metrics, which are then used to predict the defects in the file. Shi et al. (Shi, Lu, Chang, & Wei, 2020) also propose a model for defect prediction based on Bi-LSTM network

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

and AST path pair representation.

Convolutional Neural Networks are utilized to process data with a grid-like architecture (Goodfellow, Bengio, & Courville, 2016). This network uses a mathematical process called convolution instead of standard matrix multiplication. CNN mainly consists of three layers: 1) Convolutional layers; 2) nonlinear layers; 3) pooling layers. Many researchers explored CNN networks in their studies; for example, Li et al. (J. Li, et al., 2017) proposed an SDP model based on CNN, named DP-CNN. Their approach is based on the programs' ASTs; token vectors are extracted and encoded as numerical values. They feed these numerical vectors into CNN to learn the semantic information of programs. After that, the semantic features are used for defect prediction. Hoang et al. (Hoang, Dam, Kamei, Lo, & Ubayashi, 2019) proposed a deep learning framework called DeepJIT. The CNN is the basis for this model. It employs convolutional network layers to process commits, text, and code changes, and a combination layer combines the two embedding vectors into one. Meilong et al. (Meilong, He, Xiao, Li, & Zeng, 2020) also presented a CNN model to learn source code semantic features and then use these semantic features for predicting defects.

Transformer models are also used to represent source code and predict software defects, Guo et al. (Guo, et al., 2020) presented GraphCodeBERT, a multilayer transformer framework. It uses three main components as input: source code, data flow graph, and paired comments. It assists with code clone identification, translation, refinement, and other downstream code-related processes.

More defect prediction methods based on semantic features using deep learning models were proposed; Qiao et al. (Qiao, Li, Umer, & Guo, 2020) introduced a new method to predict the fault count in software systems using deep learning techniques. They started by performing data normalization on an available dataset. They are then modeled the data to get it ready for the DL model. Finally, they fed the processed data into a neural network to predict the fault count. Tong et al. (Tong, Liu, & Wang, 2018) introduced SDAEsTSE, a new SDP approach that combines SDAEs and ensemble learning, especially the suggested two-stage ensemble learning (TSE). The two main phases of their method are the deep learning phase and TSE phase. They first employ SDAEs to extract DPs from standard software features and then propose TSE as a novel ensemble learning strategy to address the problem of class imbalance. Zhao et al. (Zhao, et al., 2019) introduced a defect prediction model called Siamese parallel fully connected network, which combines the benefits of deep learning and Siamese networks into a single method. The AdamW method is used to train this model and determine the best weights.

Zhou et al. (Zhou, et al., 2019) proposed a new SDP model based on deep forest (DPDF). Deep learning and ensemble learning are fully utilized in this design. This model detects more significant defect features using a new cascade method, transforming random forest classifiers into a layer-by-layer design. Fan et al. (Fan, Diao, Yu, Yang, & Chen, 2019) proposed an SDP model using attention-based RNN. Their model generates semantic information from the source code and then uses them in predicting defects. Xu et al. (J. Xu, Wang, & Ai, 2020) provided a model for predicting software defects based on graph neural networks (GNN-DP),

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

detecting software defect patterns that combine semantics and context information. GNNs are used in GNN-DP to collect semantic information to predict defective modules. In addition, their approach combines code concepts and semantic information to capture defect patterns in specific contexts.

6) CHALLENGES AND FUTURE DIRECTIONS IN SOFTWARE DEFECTS PREDICTION

Based on the results of the previous and current studies, in this part, we will discuss some challenges and future trends in software defect prediction.

- **Semantics information of the source code and unlabeled dataset.**

Nowadays, deep learning models can process long texts, such as source code. Unlike handcrafted features, source code semantics features are collected bottom-up, including semantics information and the source code structure. As a result, the dimension of labeled datasets will be reduced; and the unlabeled data will be used to pretrain the associated activities, allowing trained models to grasp the source code more deeply and thoroughly. This procedure will enable the detection of more profound defects.

- **Defect prediction in mobile applications.**

Mobile applications have received widespread attention because of their characteristics of simplicity, portability, timeliness, and efficiency. The business community is constantly developing office applications and mobile applications suitable for various application scenarios. Therefore, the importance of mobile applications is self-evident. The type of device on which the application is installed and used, the size and application complexity, and the operating system are significant differences between mobile and desktop applications. Current defect prediction methods mainly focus on desktop applications, and it is still unknown whether these methods apply to mobile applications. Therefore, it is necessary further to explore the applicability of existing approaches in mobile applications.

- **Study severity of software defect reports.**

At present, most defect prediction models give whether a module has defects. Still, there are few studies on the number of defects in a specific module and the severity of the defect severity of the module. Further research on a range of classification and prediction challenges is required to perform systematic analysis and research on the severity of software defect reports. Thus, researchers should give a lot of attention and focus on the accurate, comprehensive, and efficient evaluation of the severity of defect reports in the maintenance of large open-source projects.

- **Cross-project defect prediction in closed source software projects.**

In cross-project SDP, various new techniques proposed by researchers are based on open-source software projects. However, there are significant distinctions between open and closed source software (such as commercial software). For example, the details of code

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

implementation in closed source software are not disclosed. Therefore, whether the existing cross-project defect prediction methods are suitable for closed-source software projects remains explored.

7) CONCLUSION

In software engineering, SDP is a significant challenge. Defect prediction models can enhance the quality of software systems while also decreasing the cost of delivery. Early in the software lifecycle, using SDP models allows developers to focus their testing efforts such that parts classified as "prone to defects" are checked more thoroughly than other parts of the software project. As a result, various SDP studies have been published. According to our review of more than 165 previous studies, some have provided promising results helpful in this field. Still, many were less valuable than they could be and offered insufficient contextual information to fully comprehend the model.

This review also presents the recent research progress in SDP using deep learning algorithms. We describe the primary challenges of the defect prediction issue as code representation and a lack of data, and we explore how to deal with these issues. A defect is usually not specific to a small function or line of code, and it can be fixed in various ways. For example, a defect can be corrected either inside the code or when calling it. As a result, the fault no longer has exact coordinates inside this source file. Furthermore, if a line of code is not an explicit defect, it can become defective over time. The purpose of the code may change due to the changed context, and the previous implementation may no longer match the modern requirements or specifications.

References

1. Abdu, A., Zhai, Z., Algabri, R., Abdo, H. A., Hamad, K., & Al-antari, M. A. (2022). Deep Learning-Based Software Defect Prediction via Semantic Key Features of Source Code—Systematic Survey. Mathematics, 10, 3120.
2. Akimova, E. N., Bersenev, A. Y., Deikov, A. A., Kobylkin, K. S., Konygin, A. V., Mezentsev, I. P., & Misilov, V. E. (2021). A Survey on Software Defect Prediction Using Deep Learning. Mathematics, 9, 1180.
3. Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 51, 1-37.
4. Allen, F. E. (1970). Control flow analysis. ACM Sigplan Notices, 5, 1-19.
5. Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., & Bier, L. (1998). Clone detection using abstract syntax trees. In Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272) (pp. 368-377): IEEE.
6. Boehm, B., & Basili, V. R. (2007). Software defect reduction top 10 list. Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research, 34, 75.
7. Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., & Panichella, S. (2013). Multi-objective cross-project defect prediction. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (pp. 252-261): IEEE.
8. Canfora, G., Lucia, A. D., Penta, M. D., Oliveto, R., Panichella, A., & Panichella, S. (2015). Defect prediction

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

as a multiobjective optimization problem. Software Testing, Verification and Reliability, 25, 426-459.

9. Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. Expert Systems with Applications, 36, 7346-7354.
10. Chen, J., Hu, K., Yang, Y., Liu, Y., & Xuan, Q. (2020). Collective transfer learning for defect prediction. Neurocomputing, 416, 103-116.
11. Chen, L., Fang, B., Shang, Z., & Tang, Y. (2015). Negative samples reduction in cross-company software defects prediction. Information and Software Technology, 62, 67-77.
12. Cruz, A. E. C., & Ochimizu, K. (2009). Towards logistic regression models for predicting fault-prone code across software projects. In 2009 3rd international symposium on empirical software engineering and measurement (pp. 460-463): IEEE.
13. Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. Information Sciences, 264, 260-278.
14. D'Ambros, M., Lanza, M., & Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering, 17, 531-577.
15. Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2021). Within-project defect prediction of infrastructure-as-code using product and process metrics. IEEE Transactions on Software Engineering, 1-1.
16. Dam, H. K., Pham, T., Ng, S. W., Tran, T., Grundy, J., Ghose, A., Kim, T., & Kim, C.-J. (2018). A deep tree-based model for software defect prediction. arXiv preprint arXiv:1802.00921.
17. Dam, H. K., Pham, T., Ng, S. W., Tran, T., Grundy, J., Ghose, A., Kim, T., & Kim, C.-J. (2019). Lessons learned from using a deep tree-based model for software defect prediction in practice. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (pp. 46-57): IEEE.
18. Deng, J., Lu, L., & Qiu, S. (2020). Software defect prediction via LSTM. IET software, 14, 443-450.
19. Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 81, 649-660.
20. Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019). Software defect prediction via attention-based recurrent neural network. Scientific Programming, 2019.
21. Felix, E. A., & Lee, S. P. (2017). Integrated approach to software defect prediction. IEEE Access, 5, 21524-21547.
22. Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. IEEE Transactions on Software Engineering, 25, 675-689.
23. Fenton, N. E., & Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software Engineering, 26, 797-814.
24. Gabel, M., Jiang, L., & Su, Z. (2008). Scalable detection of semantic clones. In Proceedings of the 30th international conference on Software engineering (pp. 321-330).
25. Gong, L., Jiang, S., Bo, L., Jiang, L., & Qian, J. (2019). A novel class-imbalance learning approach for both within-project and cross-project defect prediction. IEEE Transactions on Reliability, 69, 40-54.
26. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning: MIT press.
27. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., & Fu, S. (2020). Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366.
28. Gyimóthy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transactions on Software Engineering, 31, 897-910.

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

29. Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38, 1276-1304.
30. Hassan, A. E. (2009). Predicting faults using the complexity of code changes. In 2009 IEEE 31st international conference on software engineering (pp. 78-88): IEEE.
31. He, P., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59, 170-190.
32. He, P., Li, B., & Ma, Y. (2014). Towards cross-project defect prediction with imbalanced feature sets. arXiv preprint arXiv:1411.4228.
33. He, P., Li, B., Zhang, D., & Ma, Y. (2014). Simplification of training data for cross-project defect prediction. arXiv preprint arXiv:1405.0773.
34. He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering, 19, 167-199.
35. Herbold, S. (2013). Training data selection for cross-project defect prediction. In Proceedings of the 9th international conference on predictive models in software engineering (pp. 1-10).
36. Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18, 1527-1554.
37. Hoang, T., Dam, H. K., Kamei, Y., Lo, D., & Ubayashi, N. (2019). DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (pp. 34-45): IEEE.
38. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9, 1735-1780.
39. Hosseini, S., Turhan, B., & Gunarathna, D. (2017). A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering, 45, 111-147.
40. Hua, W., Sui, Y., Wan, Y., Liu, G., & Xu, G. (2020). FCCA: Hybrid code representation for functional clone detection using attention networks. IEEE Transactions on Reliability, 70, 304-318.
41. Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. Empirical Software Engineering, 13, 561-595.
42. Jiang, Y., Cukic, B., & Menzies, T. (2008). Can data transformation help in the detection of fault-prone modules? In Proceedings of the 2008 workshop on Defects in large software systems (pp. 16-20).
43. Jin, C. (2011). Software reliability prediction based on support vector regression using a hybrid genetic algorithm and simulated annealing algorithm. IET software, 5, 398-405.
44. Jin, C. (2021a). Cross-project software defect prediction based on domain adaptation learning and optimization. Expert Systems with Applications, 171, 114637.
45. Jin, C. (2021b). Software defect prediction model based on distance metric learning. Soft Computing, 25, 447-461.
46. Jin, C., & Jin, S.-W. (2015). Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. Applied Soft Computing, 35, 717-725.
47. Jing, X.-Y., Wu, F., Dong, X., & Xu, B. (2016). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Transactions on Software Engineering, 43, 321-339.
48. Jing, X., Wu, F., Dong, X., Qi, F., & Xu, B. (2015). Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In Proceedings of the 2015 10th Joint

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

Meeting on Foundations of Software Engineering (pp. 496-507).

49. Jureczko, M., & Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. In Proceedings of the 6th international conference on predictive models in software engineering (pp. 1-10).
50. Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., & Hassan, A. E. (2016). Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 21, 2072-2106.
51. Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2012). A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 39, 757-773.
52. Kamiya, T., Kusumoto, S., & Inoue, K. (2002). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 28, 654-670.
53. Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. Information and Software Technology, 58, 388-402.
54. Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering, 34, 485-496.
55. Li, J., He, P., Zhu, J., & Lyu, M. R. (2017). Software defect prediction via convolutional neural network. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 318-328): IEEE.
56. Li, M., Zhang, H., Wu, R., & Zhou, Z.-H. (2012). Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 19, 201-230.
57. Li, N., Shepperd, M., & Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 122, 106287.
58. Li, Z., Jing, X.-Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. IET software, 12, 161-175.
59. Liang, H., Yu, Y., Jiang, L., & Xie, Z. (2019). Seml: A semantic LSTM model for software defect prediction. IEEE Access, 7, 83812-83824.
60. Limsettho, N., Bennin, K. E., Keung, J. W., Hata, H., & Matsumoto, K. (2018). Cross project defect prediction using class distribution estimation and oversampling. Information and Software Technology, 100, 87-102.
61. Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. Information and Software Technology, 107, 125-136.
62. Lu, H., Kocaguneli, E., & Cukic, B. (2014). Defect prediction between software versions with active learning and dimensionality reduction. In 2014 IEEE 25th International Symposium on Software Reliability Engineering (pp. 312-322): IEEE.
63. Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. Information and Software Technology, 54, 248-256.
64. Majd, A., Vahidi-Asl, M., Khalilian, A., Poorsarvi-Tehrani, P., & Haghighi, H. (2020). SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications, 147, 113156.
65. Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27, 504-518.
66. Meilong, S., He, P., Xiao, H., Li, H., & Zeng, C. (2020). An approach to semantic and structural features

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

learning for software defect prediction. Mathematical Problems in Engineering, 2020.

67. Menzies, T., Dekhtyar, A., Distefano, J., & Greenwald, J. (2007). Problems with Precision: A Response to" comments on'data mining static code attributes to learn defect predictors'". IEEE Transactions on Software Engineering, 33, 637-640.
68. Mizuno, O., & Hirata, Y. (2014). A cross-project evaluation of text-based fault-prone module prediction. In 2014 6th International Workshop on Empirical Software Engineering in Practice (pp. 43-48): IEEE.
69. Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of the 30th international conference on Software engineering (pp. 181-190).
70. Nagappan, N., Ball, T., & Zeller, A. (2006). Mining metrics to predict component failures. In Proceedings of the 28th international conference on Software engineering (pp. 452-461).
71. Nam, J. (2014). Survey on software defect prediction. Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep.
72. Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2017). Heterogeneous defect prediction. IEEE Transactions on Software Engineering, 44, 874-896.
73. Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. In 2013 35th international conference on software engineering (ICSE) (pp. 382-391): IEEE.
74. Ni, C., Liu, W.-S., Chen, X., Gu, Q., Chen, D.-X., & Huang, Q.-G. (2017). A cluster based feature selection method for cross-project software defect prediction. Journal of Computer Science and Technology, 32, 1090-1107.
75. Pan, C., Lu, M., Xu, B., & Gao, H. (2019). An improved CNN model for within-project software defect prediction. Applied Sciences, 9, 2138.
76. Panichella, A., Oliveto, R., & De Lucia, A. (2014). Cross-project defect prediction models: L'union fait la force. In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) (pp. 164-173): IEEE.
77. Peters, F., & Menzies, T. (2012). Privacy and utility for defect prediction: Experiments with morph. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 189-199): IEEE.
78. Peters, F., Menzies, T., Gong, L., & Zhang, H. (2013). Balancing privacy and utility in cross-company defect prediction. IEEE Transactions on Software Engineering, 39, 1054-1068.
79. Peters, F., Menzies, T., & Layman, L. (2015). LACE2: Better privacy-preserving data sharing for cross project defect prediction. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 801-811): IEEE.
80. Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. Neurocomputing, 385, 100-110.
81. Rahman, F., Posnett, D., & Devanbu, P. (2012). Recalling the" imprecision" of cross-project defect prediction. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (pp. 1-11).
82. Rathore, S. S., & Kumar, S. (2019). A study on software fault prediction techniques. Artificial Intelligence Review, 51, 255-327.
83. Ryu, D., Choi, O., & Baik, J. (2016). Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empirical Software Engineering, 21, 43-71.
84. Ryu, D., Jang, J.-I., & Baik, J. (2015). A hybrid instance selection using nearest-neighbor for cross-project

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

defect prediction. Journal of Computer Science and Technology, 30, 969-980.

85. Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K., & Lopes, C. V. (2016). Sourcerercc: Scaling code clone detection to big-code. In Proceedings of the 38th International Conference on Software Engineering (pp. 1157-1168).
86. Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Folleco, A. (2014). An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Information Sciences, 259, 571-595.
87. Seliya, N., Khoshgoftaar, T. M., & Zhong, S. (2005). Analyzing software quality with limited fault-proneness defect data. In Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05) (pp. 89-98): IEEE.
88. Shi, K., Lu, Y., Chang, J., & Wei, Z. (2020). PathPair2Vec: An AST path pair-based code representation method for defect prediction. Journal of Computer Languages, 59, 100979.
89. Subramanyam, R., & Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on Software Engineering, 29, 297-310.
90. Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Information and Software Technology, 96, 94-111.
91. Tran, H. D., Hanh, L. T. M., & Binh, N. T. (2019). Combining feature selection, feature learning and ensemble learning for software fault prediction. In 2019 11th International Conference on Knowledge and Systems Engineering (KSE) (pp. 1-8): IEEE.
92. Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14, 540-578.
93. Turhan, B., Mısırlı, A. T., & Bener, A. (2013). Empirical evaluation of the effects of mixed project data on learning defect predictors. Information and Software Technology, 55, 1101-1118.
94. Uchigaki, S., Uchida, S., Toda, K., & Monden, A. (2012). An ensemble approach of simple regression models to cross-project fault prediction. In 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (pp. 476-481): IEEE.
95. Wahono, R. S. (2015). A systematic literature review of software defect prediction. Journal of Software Engineering, 1, 1-16.
96. Wan, Z., Xia, X., Hassan, A. E., Lo, D., Yin, J., & Yang, X. (2018). Perceptions, expectations, and challenges in defect prediction. IEEE Transactions on Software Engineering, 46, 1241-1266.
97. Wang, H., Zhuang, W., & Zhang, X. (2021). Software Defect Prediction Based on Gated Hierarchical LSTMs. IEEE Transactions on Reliability, 70, 711-727.
98. Wang, S., Liu, T., Nam, J., & Tan, L. (2018). Deep semantic feature learning for software defect prediction. IEEE Transactions on Software Engineering, 46, 1267-1293.
99. Wang, S., Liu, T., & Tan, L. (2016). Automatically learning semantic features for defect prediction. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) (pp. 297-308): IEEE.
100. Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability, 62, 434-443.
101. Wei, H., Hu, C., Chen, S., Xue, Y., & Zhang, Q. (2019). Establishing a software defect prediction model via effective dimension reduction. Information Sciences, 477, 399-409.
102. Wu, F., Jing, X.-Y., Sun, Y., Sun, J., Huang, L., Cui, F., & Sun, Y. (2018). Cross-project and within-project semisupervised software defect prediction: A unified approach. IEEE Transactions on Reliability, 67, 581-597.

The Seybold Report
ISSN 1533-9211
DOI 10.5281/zenodo.7318104

103. Wu, R., Zhang, H., Kim, S., & Cheung, S.-C. (2011). Relink: recovering links between bugs and changes. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (pp. 15-25).

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

105. Xu, J., Wang, F., & Ai, J. (2020). Defect Prediction With Semantics and Context Features of Codes Based on Graph Representation Learning. IEEE Transactions on Reliability, 70, 613-625.

106. Xu, Z., Pang, S., Zhang, T., Luo, X.-P., Liu, J., Tang, Y.-T., Yu, X., & Xue, L. (2019). Cross project defect prediction via balanced distribution adaptation based transfer learning. Journal of Computer Science and Technology, 34, 1039-1062.

107. Yang, X., Lo, D., Xia, X., Zhang, Y., & Sun, J. (2015). Deep learning for just-in-time defect prediction. In 2015 IEEE International Conference on Software Quality, Reliability and Security (pp. 17-26): IEEE.

108. Yang, X., Yu, H., Fan, G., Shi, K., & Chen, L. (2019). Local versus global models for just-in-time software defect prediction. Scientific Programming, 2019.

109. Yatish, S., Jiarpakdee, J., Thongtanunam, P., & Tantithamthavorn, C. (2019). Mining software defects: should we consider affected releases? In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 654-665): IEEE.

110. Yousefi, J., Sedaghat, Y., & Rezaee, M. (2015). Masking wrong-successor Control Flow Errors employing data redundancy. In 2015 5th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 201-205): IEEE.

111. Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y. (2016). Towards building a universal defect prediction model with rank transformed predictors. Empirical Software Engineering, 21, 2107-2145.

112. Zhang, F., Zheng, Q., Zou, Y., & Hassan, A. E. (2016). Cross-project defect prediction using a connectivity-based unsupervised classifier. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) (pp. 309-320): IEEE.

113. Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 783-794): IEEE.

114. Zhang, Y., Lo, D., Xia, X., & Sun, J. (2015). An empirical study of classifier combination for cross-project defect prediction. In 2015 IEEE 39th Annual Computer Software and Applications Conference (Vol. 2, pp. 264-269): IEEE.

115. Zhao, L., Shang, Z., Zhao, L., Zhang, T., & Tang, Y. Y. (2019). Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks. Neurocomputing, 352, 64-74.

116. Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. Information and Software Technology, 114, 204-216.

117. Zhu, K., Ying, S., Zhang, N., & Zhu, D. (2021). Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. Journal of Systems and Software, 111026.

118. Zhu, K., Zhang, N., Ying, S., & Wang, X. (2020). Within-project and cross-project software defect prediction based on improved transfer naive bayes algorithm. Computers, Materials and Continua, 63, 891-910.

119. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., & Murphy, B. (2009). Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 91-100).