2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

Assessing the Significant Impact of Concept Drift in Software Defect Prediction

Md Alamgir Kabir*, Jacky W. Keung*, Kwabena E. Bennin† and Miao Zhang*
*Department of Computer Science, City University of Hong Kong, Hong Kong
†Department of Software Engineering, Blekinge Institute of Technology, Sweden
{makabir4-c, miazhang9-c}@my.cityu.edu.hk, Jacky.Keung@cityu.edu.hk, kwabena.ebo.bennin@bth.se

***Abstract*—*Concept drift* is a known phenomenon in software data analytics. It refers to the changes in the data distribution over time. The performance of analytic and prediction models degrades due to the changes in the data over time. To improve prediction performance, most studies propose that the prediction model be updated when *concept drift* occurs. In this work, we investigate the existence of *concept drift* and its associated effects on software defect prediction performance. We adopt the strategy of an empirically proven method DDM (Drift Detection Method) and evaluate its statistical significance using the chi-square test with Yates continuity correction. The objective is to empirically determine the *concept drift* and to calibrate the base model accordingly. The empirical study indicates that the *concept drift* occurs in software defect datasets, and its existence subsequently degrades the performance of prediction models. Two types of *concept drifts* (gradual and sudden drifts) were identified using the chi-square test with Yates continuity correction in the software defect datasets studied. We suggest *concept drift* should be considered by software quality assurance teams when building prediction models.**

***Keywords*-concept drift detection; defect prediction; empirical software engineering; software quality; streaming data**

I. Introduction

Software defect prediction models are developed from historical software data to improve the quality by fixing the bugs of the software [1], [2]. These types of models have been developed over the years to identify bugs in the source code to reduce the development cost [9]. If the historical data (i.e., fixed distributions) changes over time, the prediction model becomes outdated [6]. Krishna and Menzies [2] caution that if the prediction model changes because of the changing historical data, managers will have less faith in the potential of the trained prediction model leading to inefficient use of scarce testing resources.

The characteristics of data distribution changes over time i.e. referred to as *concept drift*. Recently, several defect prediction models based on historical data (fixed distribution) [1]–[4], [6]–[8], [10] have been proposed. These defect datasets however do change over time [2]. Krishna and Menzies studied over 15 chronological datasets where the accuracy of defect prediction model varied [2]. According to Dong et al. [6], a well-trained prediction model could give more inaccurate results if the incoming data starts to drift over time. Inconsistencies between the historical data and current incoming data thus make the model training process more challenging [5], [8] and complicated [7]. This leads to wrong prediction which disrupts the goal of achieving quality software [2].

Several software defect prediction models have been developed considering the feature selection methods, data preprocessing techniques, and sampling strategy [1] on the C&K (Chidamber & Kemerer object-oriented) [31] and process metrics datasets [33]. The impact of the data distribution changes over time on prediction performance has not been considered by many. Few studies [11]–[13] observed that prediction accuracy fluctuates when chronological datasets are used for model training. They attributed the fluctuating results to changes in the data features. The first to investigate the notion of *concept drift* in software defect prediction was Ekanayake et al. [12], [13]. By examining the bug fixing process and files of software programmers, they observed that changes (fixes) to software files tend to change the concept in data distribution [13] consequently impacting the performance of defect predictions.

According to *concept drift* literature, it is a very complex task to learn due to the streaming data [14]. There have been developed methods to detect the *concept drift* over the last years to understand when and how the data starts to drift that affects the performance of prediction models [15], [16]. The studies by Ekanayake et al. [12], [13] provide the foundation to investigate the phenomenon of *concept drift* in software defect prediction studies and to examine and assess the impact of *concept drift* on prediction performance. The studies of Ekanayake et al. [12], [13] are limited and not comprehensive considering most defect prediction models are built using static code metrics. Additionally, evaluation of the statistical significance of *concept drift* effects on prediction performance using robust statistical tests are yet to be done. To the best of our knowledge, this is the first attempt to identify drift in software engineering domain (i.e. detecting the defect in the source code where the target variable is defect) by applying *concept drift* detection method and validating with statistical test. In this paper, we conduct empirical experiments to detect drift in the chronological defect datasets by adopting the strategy of drift detection method, DDM [15] and to evaluate its statistical significance using the chi-square test with the Yates continuity correction [17]. In conclusion, we investigate the existence and types of *concept drift* in software defect datasets. We observe the existence of two types of *concept drift* (gradual and

978-1-7281-2607-4/19/$31.00 ©2019 IEEE
DOI 10.1109/COMPSAC.2019.00017

sudden) in defect data and the significance of *concept drift* in software defect prediction.

The remaining parts of this paper are structured as follows. Section II discusses the related studies. In section III, we discuss the learning process under streaming data by defining *concept drift* and drift detection model. In section IV, we describe our experimental methodology. Section V presents the empirical results and discussion. Threats to validity of this empirical study are presented in Section VI. Finally, section VII concludes the paper with the summary of our work and future directions.

II. Related Work

Several studies have been conducted in the domain of Software Defect Prediction (SDP). Most often, researchers are concerned with the prediction performance instability in SDP. However, this study reports the existence of *concept drift* that affects performance of defect prediction models.

Bernstein et al. [11] argued that prediction performance depends on the temporal features of the data. They conducted an experiment by using non-linear models to find the hidden relationship between the features of the data. After their first experiment [11], Ekanayake et al. [12] extended their previous study and investigated why the prediction quality varies in time in terms of accuracy. They increased their experimental datasets by extracting the Mozilla, Open Office, and Netbeans open source projects. Their experiment illustrates that the concept changes due to the changing of the bug fixing process. Furthermore, Ekanayake et al. [13] investigated the instability of the accuracy of the defect prediction model by observing the features of the projects that cause the changes of the quality of prediction model. In their experiments, they observed that changing the distributions affects the quality of the prediction model. Note that these studies [11]–[13] investigated the reason why the quality of the prediction model changes over time and to identify the stability of the prediction model on different versions of open source bug datasets. These datasets are formed by extracting the data features such as versioning system features. These previous works examined the open source projects features to find the reasons behind the stability and variability of the accuracy of the prediction models as well. By conducting their experiments, the authors got the notion of *concept drift* in software projects affecting the prediction quality. However, these studies [11]–[13] recommend the need to conduct an intensive investigation by utilizing the drift detection method on those datasets. Our study differentiates itself from previous studies [11]–[13] in that, we investigate the C&K metrics-based defect datasets to identify drifts and the types of drifts by adopting the strategy of the drift detection method, DDM and to evaluate its statistical significance by using chi-square test with the Yates Continuity Correction.

III. Background

A. Concept Drift

A formal definition of *concept drift* refers to the changes of the concept of the data over time. According to Gama et al. [18], the changes of classification (i.e., stream classification) of data over time refers to *concept drift*. As an example, the class distribution, A of the distribution, D changes in the time t and u and *concept drift* can be defined as $D_t(A) \neq D_u(A)$. The probabilistic definition of *concept drift* refers to the changes in probabilities characteristics of the distributions over time.

B. Drift Types

Four (4) types of *concept drift* have been identified in the literature [18]–[20]. These are sudden drift, gradual drift, incremental drift, and recurring concepts. According to Gama et al. [15], at the time t, if the concept changes suddenly or replaced suddenly from C_i to C_j, it is referred to as a sudden drift. In other words, sudden drift refers to *concept drift* over a short time. For the gradual drift, the source is shifted gradually which means gradually replaces the old concept by the new one. Incremental drift happens by changing the instances very incrementally. The changes between the sources are very little. This type of drift is observed in a longer period. The reoccurring concepts means the old active concept appears again after certain period. Fig. 1 shows the types of *concept drift*.

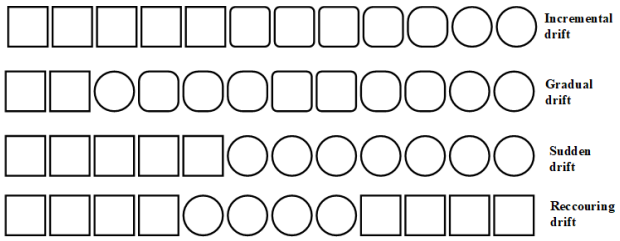

Figure 1. Different types of drift [19]

C. Concept Drift Detection Method

There have been developed many techniques to deal with *concept drift* [18]–[20]. The most typical techniques are the following: (a) window-based technique, (b) weight-based technique, and (c) techniques of ensemble classifiers. For training a classifier, window-based techniques select previous datasets. Lu et al. conduced an experiment to observe the significance effects in the prediction model [14]. It is also reported by the literature that the window size is determined by the user [14]. In the window-based techniques, memory-based approaches are the following: full-memory, no-memory, and size of the window approach [14]. In Fig. 2, the learner is trained on the old data that cannot forget. This approach is called full-memory based approach where the window sized is fixed by the user [14].

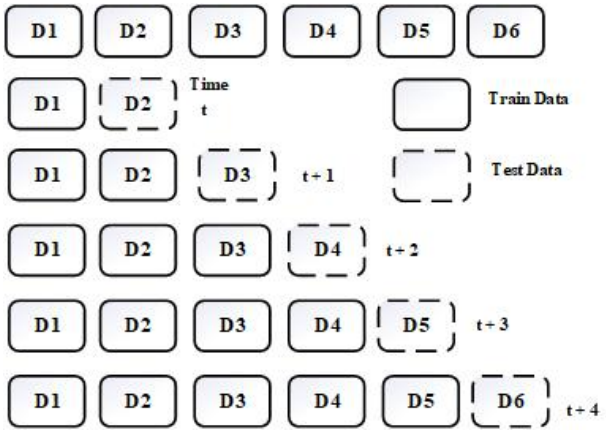

Figure 2. Full memory-based approach [14]

1) Drift Detection Method (DDM): DDM, one of the referred and well-known drift detection method introduced by Gama et al. [15]. This method is based on the full-memory based approach. According to Lu et al. [14], the window is defined by the user in this method. This method is also called full memory window-based drift detection method and model is trained on the old data [14]. Gama et al. [15] monitored the error-rate of the learning algorithm. They claimed that if the error-rate of the learning algorithms decreases, the distribution is stationary. If the error-rate of the learning algorithms increases, the data distribution becomes non-stationary. They introduced the warning level to detect drift in streaming data. If the error-rate increases, the data distribution changes and a warning level is declared. If changes are detected by the warning level, a new learner is built. DDM works on the two-time windows of instances reported by Lu et al. [14] and Gama et al. [15].

IV. Experimental Settings

In this section, we discuss the datasets used in this experiment which are publicly available in the SeaCraft repository [35]. Experiments are conducted using the statistical tool R [25] using the packages, Caret [26] and e1071 [27].

A. Datasets

Twenty-eight (28) versions of the open source project datasets denoted by Menzies et al. [21] and Marian Jureckzo [22] that were collected from the SeaCraft repository [35]. A general summary of our selected datasets is presented in Table I. These datasets are comprised of C&K metrics which are widely used in in software defect prediction [31]. More explanations of the metrics could be found in [33].

B. Evaluation Measures

In this study, the error-rate is computed to assess the performance of the learners and monitor the fluctuation to detect the changes in data distributions. The mathematical

Table I
Summary of Studied 28 Datasets Collected From Seacraft Repository

Studied Dataset	Project	Version	# of Modules	# of Faulty Modules	% Faulty Modules
NASA	jm1	1	7782	1672	21.49%
		1.2	9593	1759	18.34%
		1.3	7782	1672	21.49%
Jureczko	prop	4	3022	264	8.74%
		9	4255	149	3.50%
		40	4053	466	11.50%
		44	4620	389	8.42%
		85	3077	948	30.81%
		92	3557	1269	35.68%
		121	2998	425	14.18%
		128	3527	220	6.24%
		157	2496	367	14.70%
		164	3457	319	9.23%
		185	2825	268	9.49%
		192	3598	85	2.36%
		225	1810	147	8.12%
		236	2231	76	3.41%
		245	1962	103	5.25%
		256	1964	625	31.82%
		285	1694	177	10.45%
		292	2285	209	9.15%
		305	2344	89	3.80%
		318	2395	365	15.24%
		347	2871	162	5.64%
		355	2791	924	33.11%
		362	2854	213	7.46%
		452	256	33	12.89%
		453	192	20	10.42%

definition of this measures is presented as $Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$ where $Error - rate = 1 - Accuracy$. From the predictive class level, true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are computed to calculate error-rate. The detailed description of this measure can be found in [34].

C. Statistical Test

Chi-square test with Yates continuity correction is adopted to discover the existence of *concept drift* and to detect the statistically significant difference between two distributions that change over time. This statistical test is mostly used in different domains to detect the significant changes of data distributions and recommended by [14]–[17] for comparing two data distributions for identifying the drift data point. The statistically significance value is computed using = 0:05. Nishida et al. [17] conducted an experiment for detecting *concept drift* in two-time window-based data distributions named as STEPD (Statistical Test of Equal Proportions Detection) where the warning level ($p - value$) for drift detection was defined as 0.05 (i.e., the observed significance level) [17]. This determined significance level was used as the warning level in our experiment and used by the other [14], [30] drift detection methods. For all the experiments, null hypothesis (H0) is that there is no difference between two distributions where the significance level is 0.05. If the observed significance level is less than the significance level

(0.05), the null hypothesis (H0) is rejected and the alternative hypothesis (H1) is accepted that is to say, *concept drift* is detected. The test statistic is computed as follows:

$$T(CC_o, CC_r, OD_o, RD_n) = \frac{|a| - 0.5b}{\sqrt{p(1-p)b}} \quad (1)$$

The equations are an exception to the prescribed spe where, $a = |CC_o/OD_o - CC_r/RD_r|$, $b = (1/OD_o + 1/RD_r)$, and $p = (CC_o + CC_r)/(OD_o + RD_r)$. In equation 1,the CC_o is the number of correct classifications among the overall, OD_o distributions and CC_r = number of correct classifications among the recent, RD_r distributions.

D. Experimental Procedure

This experiment reports the existence and types of *concept drift* in software defect datasets. Two defect datasets (*jm1* and *prop*) are used in this experiment. The *jm1* dataset includes three versions from which 500 instances are initially used in the first experimental run and increased by 500 instances for subsequent runs. For each run, the model is trained on the data by taking 20% as the training set and remaining data is used as test data to compute the performance measures. Afterwards, another batch of 500 instances are added to the old (previous) instances and the same procedure is implemented to build the model to compute the error-rate. This process is iterated until all data instances have been used for this experiment. We select Naive Bayes and Decision Tree classifier to produce the error-rate. These learners have been reported as good in the study of defect prediction by Agrawal and Menzies [28] and in the *concept drift* literature [29]. The prediction models are built at each experimental run to produce the error-rate values. The error-rate values are monitored to detect changes (significant increases or decreases of error-rate) for drift detection similar to the drift detection approaches by [14], [29], [32]. By using the equation 1, the $p-values$ are calculated for each trial to find the statistical significance. From the prop datasets, 25 versions are included in our experiments as described in the Table I. The first version (oldest) is used for the first run and subsequent versions are added to this version (oldest) for subsequent runs. The procedure is iterated until all versions of the datasets have been exhausted. Significant changes are monitored in error-rates to detect changes in the distributions.

V. Experimental Results and Discussions

This experiment investigates the existence of *concept drift* and its significance in our considered software defect datasets using a drift detection method, DDM. We monitor the error-rate of every experimental run. For statistically significant results, Table II and III presents $p - values$ obtained after conducting the statistical test for the *jm1* and *prop* datasets.

Table II
Performance Observation to Detect the *Concept Drift* Across Three (3) Versions of the *jm1* Datasets

# of Overall Data (OD_o)	Recent Data (RD_r)	Observed Version	Naive Bayes	Decision Tree
Historical Data	New data		$p - value$	$p - value$
500	-	-	-	-
1000	500	1	0.249	0.747
1500	500	1	0.296	0.475
2000	500	1	0.717	0.603
2500	500	1.2	0.102	0.078
3000	500	1.2	**0.027**	**0.025**
3500	500	1.2	0.121	0.222
4000	500	1.3	**0.041**	**0.041**
4500	-	1.3	0.313	0.656

We monitor the error-rates of the overall distributions of each experimental run to monitor the changes in the next run. The significant changes of error-rate indicate the changes of concept. The concept gradually detected and replaced by the old concept in the *prop* datasets called as gradual drift. However, the bold highlighted values are the significant values in Table II and III. When we consider the Naive Bayes classifier for the *prop* datasets, we observe the significant fluctuation (i.e., in gradually) of the error-rates at the mentioned data points, 29109, 41485, 45526, 55775, and 61041. The corresponding versions of these data points are 121, 185, 225, 292 respectively and we observe maximum changes of error-rate compared to previous error-rates. The $p - values$ of these data points are 0.046, 0.032, 0.044, 0.045, and 0.046 which are also less than the significance level indicating the warning level (i.e., 0.05 [15]) of drift point.

By using the decision tree classifier, we also observe the significant fluctuation (i.e., in gradually) of the error-rates at the data points 11330, 19027, 49452, 63832 that change the concepts of data distributions. The corresponding observed versions of these data distributions are 9, 44, 245, and 347. The statistically significance values are 0.038, 0.037, 0.043, and 0.037 (Table III) which satisfy the alternative hypothesis and identify four drifting point. We discover that the error-rate of these data points fluctuates within a short period and new concept occurs for a short time for the *jm1* datasets (Table II). For statistical results, using the Nave Bayes the $p - values$ of these data points are 0.027 and 0.041, which are also less than the significance level indicating the warning level of drift point. We also observe the same fluctuation at the same data points and observed significant values are 0.025 and 0.041 which are less than warning level drift point. However, the drifting points are identified at the 3000 and 4000 data points of the version 1.2 and 1.3 of the *jm1* datasets respectively (Table II). The result suggests that the presence of *concept drifts* in the C&K metrics-based defect datasets impacts the quality of prediction. It indicates that the models need to be updated after the drift to obtain

Table III
PERFORMANCE OBSERVATION TO DETECT THE *Concept Drift* ACROSS TWENTY-FIVE (25) VERSIONS OF THE *prop* DATASETS. THE SIGNIFICANT $p - values$ ARE HIGHLIGHTED IN BOLD.

# of Overall Instances (OD_o)	# of New Instances (RD_r)	Observed Version	Naive Bayes	Decision Tree
Historical Data	New Data		$p - value$	$p - value$
3022	-	-	-	-
7277	4255	4	0.202	0.302
11330	4053	9	0.658	**0.038**
15950	4620	40	0.132	0.264
19027	3077	44	0.106	**0.037**
22584	3557	85	0.149	0.045
25582	2998	92	0.14	0.331
29109	3527	121	**0.046**	0.088
31605	2496	128	0.135	0.155
35062	3457	157	0.054	0.164
37887	2825	164	0.057	0.187
41485	3598	185	**0.032**	0.052
43295	1810	192	0.064	0.225
45526	2231	225	**0.044**	0.093
47488	1962	236	0.051	0.119
49452	1964	245	0.718	**0.043**
51146	1694	256	0.095	0.655
53431	2285	285	0.059	0.399
55775	2344	292	**0.045**	0.081
58170	2395	305	0.168	0.273
61041	2871	318	**0.046**	0.119
63832	2791	347	0.186	**0.037**
66686	2854	355	0.056	0.197
66942	256	362	0.326	0.689
67134	192	452	0.209	0.183
67346	-	453	0.134	0.303

better prediction performances. The significant $p-value$ for the corresponding data points identified in the chronological datasets that identify the *concept drift* in the versions of those datasets. It would help the SQA teams to retrain the model again to get better prediction performance. Therefore, we suggest considering *concept drift* in defect prediction for better classification.

VI. THREATS TO VALIDITY

As a preliminary study, an investigation is conducted to identify the presence of *concept drift* in software defect datasets. We discuss the possible threats to validity to our empirical results. In this experiment, we use two chronological software defect datasets, *jm1* and *prop*. These datasets are widely studied [2], [23], [24] in software defect predictions. These studied datasets are however open source projects and the use of commercial projects is left for future studies. Furthermore, empirical outcomes may not be optimistically generalized beyond these studied chronological datasets. Regarding the statistical test, chi-square test with Yates continuity correction is adopted to find the statistical significance results. This statistical test is widely recommended in the *concept drift* detection literature [15], [17], [29]. In addition, we comparatively used large datasets, since the statistical test works well on large samples [14], [15]. We consider two main learning algorithms namely Naive Bayes and Decision Tree. We choose these learners because they have been widely used in literature [23], [24], [30], [32] and they increased the prediction performances as reported by Gama et al. [15].

VII. CONCLUSION AND FUTURE WORK

Previous studies reveal that *concept drift* may occur in chronological defect datasets. However, these studies did not adopt and apply any drift detection method to identify *concept drift* in those datasets. This paper adopts a well-known and widely accepted drift detection method, DDM and investigate the existence of *concept drift* in the *jm1* and *prop* datasets. From the *jm1* datasets, two drifting points are identified in the version 1.2 and 1.3. By monitoring the error-rate of the learning algorithms, sudden drift occurred in this dataset indicating that to obtain better prediction performance, the old learner needs to be updated or in some cases replaced by a new one. From the *prop* datasets, we consider 25 versions containing 67,346 modules. Among the versions, drifts are identified in the versions of 9, 44, 121, 185, 225, 245, 292, 318, and 347 and occur gradually in these versions referred to as gradual drift. We suggest that the classification models need to be updated after the drift. This experimental study could be enhanced in several ways. As future work, we aim to extend our study by conducting the experiment on small software defect datasets with Fishers Exact test to monitor the data distributions to find the changes that effect the prediction performance.

ACKNOWLEDGMENT

This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (No.11208017) and the research funds of City University of Hong Kong (9678149, 7005028, and 9678149), and the Research Support Fund by Intel (9220097).

REFERENCES

[1] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," in IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, July-Aug. 2008.doi: 10.1109/TSE.2008.35

[2] R. Krishna and T. Menzies, "Bellwethers: A Baseline Method For Transfer Learning," in IEEE Transactions on Software Engineering.doi: 10.1109/TSE.2018.2821670

[3] T. Sethi and Gagandeep, "Improved approach for software defect prediction using artificial neural networks," 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, 2016, pp. 480-485.doi: 10.1109/ICRITO.2016.7785003

[4] X. Jing, F. Wu, X. Dong and B. Xu, "An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems," in IEEE Transactions on Software Engineering, vol. 43, no. 4, pp. 321-339, 1 April 2017.doi: 10.1109/TSE.2016.2597849

[5] J. Nam, W. Fu, S. Kim, T. Menzies and L. Tan, "Heterogeneous Defect Prediction," in IEEE Transactions on Software Engineering, vol. 44, no. 9, pp. 874-896, 1 Sept. 2018.doi: 10.1109/TSE.2017.2720603

[6] F. Dong, J. Lu, K. Li and G. Zhang, "Concept drift region identification via competence-based discrepancy distribution estimation," 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, pp. 1-7, 2017.doi: 10.1109/ISKE.2017.8258734

[7] A. Liu, Y. Song, G. Zhang, and J. Lu, "Regional concept drift detection and density synchronized drift adaptation," IJCAI Int. Jt. Conf. Artif. Intell., pp. 22802286, 2017.doi:10.24963/ijcai.2017/317

[8] Y. Song, G. Zhang, J. Lu and H. Lu, "A fuzzy kernel c-means clustering model for handling concept drift in regression," 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Naples, 2017, pp. 1-6. doi: 10.1109/FUZZ-IEEE.2017.8015515

[9] S. Wang, T. Liu, J. Nam and L. Tan, "Deep Semantic Feature Learning for Software Defect Prediction," in IEEE Transactions on Software Engineering.doi: 10.1109/TSE.2018.2877612

[10] T. Jiang, L. Tan and S. Kim, "Personalized defect prediction," 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, 2013, pp. 279-289. doi: 10.1109/ASE.2013.6693087

[11] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," Ninth Int. Work. Princ. Softw. Evol. IWPSE07, conjunction with 6th ESEC/FSE Jt. Meet., pp. 1118, 2007.doi:10.1145/1294948.1294953

[12] J. Ekanayake, J. Tappolet, H. C. Gall and A. Bernstein, "Tracking concept drift of software projects using defect prediction quality," 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, 2009, pp. 51-60.doi: 10.1109/MSR.2009.5069480

[13] J. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein, "Time variance and defect prediction in software projects: Towards an exploitation of periods of stability and change as well as a notion of concept drift in software projects," Empir. Softw. Eng., vol. 17, no. 45, pp. 348389, 2012. doi:10.1007/s10664-011-9180-x

[14] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in IEEE Transactions on Knowledge and Data Engineering.doi: 10.1109/TKDE.2018.2876857

[15] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with Drift Detection," In: Bazzan A.L.C., Labidi S. (eds) Advances in Artificial Intelligence SBIA 2004. Lecture Notes in Computer Science, pp. 286-295, 2004.doi:10.1007/978-3-540-28645-5_29

[16] M. Baena-Garca, J. del Campo-vila, R. Fidalgo, A. Bifet, R. Gavald,and R. Morales-Bueno, "Early Drift Detection Method," Evaluation, 2008

[17] K. Nishida and K. Yamauchi, "Detecting Concept Drift Using Statistical Testing," Discov. Sci. DS 2007. Lect. Notes Comput. Sci., vol. 4755, pp. 264269, 2007.doi:10.1007/978-3-540-75488-6_27

[18] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Comput Surv, Vol. 46(4), no. 44, pp. 1-37, 2014. doi:10.1145/2523813

[19] N. Lu, G. Zhang, and J. Lu, "Concept drift detection via competence models," Artif. Intell., vol. 209, no. 1, pp. 1128, 2014. doi:10.1016/j.artint.2014.01.001

[20] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," Data Min. Knowl. Discov., vol. 30, no. 4, pp. 964994, 2016.doi:10.1007/s10618-015-0448-4

[21] T. Menzies, and MartinShepperd et al. jm1 [Data set]. Zenodo, 2004.doi: 10.5281/zenodo.268514

[22] M. Jureckzo. prop [Data set]. Zenodo, 2010.doi: 10.5281/zenodo.322444

[23] J. Nam, W. Fu, S. Kim, T. Menzies and L. Tan, "Heterogeneous Defect Prediction," in IEEE Transactions on Software Engineering, vol. 44, no. 9, pp. 874-896, 1 Sept. 2018. doi: 10.1109/TSE.2017.2720603

[24] K. Muthukumaran, A. Dasgupta, and S. Abhidnya, On the Effectiveness of Cost Sensitive Neural Networks for Software Defect Prediction, vol. 614, no. SoCPaR 2016, 2018. doi:10.1007/978-3-319-60618-7_55

[25] R. C. Team, "R: A language and environment for statistical computing," Vienna, Austria: R Foundation for Statistical Computing, 2012.

[26] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt, "caret: classification and regression training . r package version 6.0-24," 2014.

[27] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel, "Misc functions of the Department of Statistics (e1071)," TU Wien. R package, 1, pp.5-24, 2008.

[28] A. Agrawal and T. Menzies, "Is "better data" better than "better data miners"?: on the benefits of tuning SMOTE for defect prediction," In Proceedings of the 40th International Conference on Software Engineering (ICSE '18). ACM, New York, NY, USA, pp.1050-1061, 2018. doi: 10.1145/3180155.3180197

[29] K. Nishida, K. Yamauchi, and T. Omori, Ace: Adaptive classifiers-ensemble system for concept-drifting environments, In: Oza N.C., Polikar R., Kittler J., Roli F. (eds) Multiple Classifier Systems. MCS 2005. Lecture Notes in Computer Science, vol 3541, pp. 176185, 2005. doi:10.1007/11494683_18

[30] L. L. Minku and X. Yao, "DDD: A New Ensemble Approach for Dealing with Concept Drift," in IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 4, pp. 619-633, April 2012. doi:10.1109/TKDE.2011.58

[31] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," in IEEE Transactions on Software Engineering, vol. 29, no. 4, pp. 297-310, April 2003. doi: 10.1109/TSE.2003.1191795

[32] Y. Hou, Dtection de concept drift, lUniversit de Technologie de Compigne, 2012

[33] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," in IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994. doi: 10.1109/32.295895

[34] K. E. Bennin, J. Keung, A. Monden, P. Phannachitta, and S. Mensah, The Significant Effects of Data Sampling Approaches on Software Defect Prioritization and Classification, Int. Symp. Empir. Softw. Eng. Meas., vol. 2017Novem, pp. 364373, 2017. doi:10.1109/ESEM.2017.50

[35] T. Menzies, R. Krishna, D. Pryor, The SEACRAFT Repository of Empirical Software Engineering Data https://zenodo.org/communities/seacraft. 2017