Cross-Version Software Defect Prediction Considering Concept Drift and Chronological Splitting

Kabir, M. A., Rehman, A. U., Islam, M. M. M., Ali, N., Baptista, M. L., & Simos, T. E. (Ed.) (2023). Cross-Version Software Defect Prediction Considering Concept Drift and Chronological Splitting. *Symmetry*, *15*(10), 1-25. Article 1934. Advance online publication. https://doi.org/10.3390/sym15101934

Link to publication record in Ulster University Research Portal

Published in:
Symmetry

Publication Status:
Published online: 18/10/2023

DOI:
10.3390/sym15101934

Document Version
Publisher's PDF, also known as Version of record

General rights
Copyright for the publications made accessible via Ulster University's Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Ulster University's institutional repository that provides access to Ulster's research outputs. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact pure-support@ulster.ac.uk.

Download date: 04/08/2024

symmetry

MDPI

Article

Cross-Version Software Defect Prediction Considering Concept Drift and Chronological Splitting

Md Alamgir Kabir [1,*], Atiq Ur Rehman [1,2], M. M. Manjurul Islam [3], Nazakat Ali [1] and Marcia L. Baptista [4]

[1] Artificial Intelligence and Intelligent Systems Research Group, School of Innovation, Design and Engineering, Mälardalen University, Högskoleplan 1, 722 20 Västerås, Sweden; nazakat.ali@mdu.se (N.A.)
[2] Department of Electrical and Computer Engineering, Pak-Austria Fachhochschule Institute of Applied Sciences and Technology, Haripur 22621, Pakistan
[3] Intelligent Systems Research Centre, University of Ulster, Londonderry BT48 7JL, UK
[4] Air Transport and Operations, Faculty of Aerospace Engineering, Delft University of Technology (TU Delft), 2628 CD Delft, The Netherlands; m.lbaptista@tudelft.nl
* Correspondence: md.alamgir.kabir@mdu.se; Tel.: +46-736620761

Abstract: Concept drift (CD) refers to a phenomenon where the data distribution within datasets changes over time, and this can have adverse effects on the performance of prediction models in software engineering (SE), including those used for tasks like cost estimation and defect prediction. Detecting CD in SE datasets is difficult, but important, because it identifies the need for retraining prediction models and in turn improves their performance. If the concept drift is caused by symmetric changes in the data distribution, the model adaptation process might need to account for this symmetry to maintain accurate predictions. This paper explores the impact of CD within the context of cross-version defect prediction (CVDP), aiming to enhance the reliability of prediction performance and to make the data more symmetric. A concept drift detection (CDD) approach is further proposed to identify data distributions that change over software versions. The proposed CDD framework consists of three stages: (i) data pre-processing for CD detection; (ii) notification of CD by triggering one of the three flags (i.e., CD, warning, and control); and (iii) providing guidance on when to update an existing model. Several experiments on 30 versions of seven software projects reveal the value of the proposed CDD. Some of the key findings of the proposed work include: (i) An exponential increase in the error-rate across different software versions is associated with CD. (ii) A moving-window approach to train defect prediction models on chronologically ordered defect data results in better CD detection than using all historical data with a large effect size ($\delta \geq 0.427$).

Keywords: chronological splitting; software defect prediction; concept drift; cross-version defect prediction

check for updates

Citation: Kabir, M.A.; Rehman, A.U.; Islam, M.M.M.; Ali, N.; Baptista, M.L. Cross-Version Software Defect Prediction Considering Concept Drift and Chronological Splitting. *Symmetry* **2023**, *15*, 1934. https://doi.org/10.3390/sym15101934

Academic Editor: Theodore E. Simos

Received: 15 August 2023
Revised: 15 October 2023
Accepted: 17 October 2023
Published: 18 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Software defect prediction (SDP) has been one of the most prominent research areas in software engineering (SE) in recent decades [1,2]. SDP helps developers to perform the early identification of faulty modules in software and assists project managers in maintaining a trade-off between quality and time-to-market [3].

In the context of prediction models, if the underlying data distributions caused by symmetric change over time for any unforeseeable reason, the performances of models based on these data also degrade [4]. We refer to this phenomenon as "concept drift", which can be formally defined as follows. Given two distinct time-points defining the time interval $[t, t+\Delta]$, if the feature vector is x and the class label is l, then the distribution is $D_{(t,\Delta)}(x,l)$. CD occurs between two distinct time-points, t and $(t+\Delta)$, $iff\ \exists\, t : D_{(t)}(x,l) \neq D_{(t+\Delta)}(x,l)$ where $D_t(x,l) = D_t(l \mid x)D_t(x)$ denotes the joint probability distribution at time t between the attributes in feature vector x and class label l.

Identifying CD is a relatively new research area in data mining and machine learning [5], but is a well-known phenomenon in data-stream mining [4]. When the underlying distribution changes over time in non-stationary environments, the data behavior is susceptible to CD. This problem was first reported as "concept drift" by Schlimmer et al. [6] who observed it while working on noisy data that were changing over time [4]. One aspect of SDP that has received very little attention is how to detect the changes across versions of chronological defect datasets. Leveraging the symmetry of the data labels can aid in building robust and generalizable models, while addressing concept drift helps ensure the model's relevance and accuracy. We discuss the existing work in this area in Section 7.

In order to construct prediction models that are more efficient, the research community should address the challenge posed by evolving data distributions over time, as elucidated by Turhan [7]. Therefore, it is necessary to be conscious of dataset shifts and their implications in the SE domain. Dataset shifts can generate biased prediction results, for example, due to changes in class prior probabilities. Changes in data probability, including probability distributions, can lead to inaccurate results, and even a well-trained prediction model will become outdated in the face of such drift, as noted by Dong et al. [8]. Furthermore, other researchers [9–13] have reported that the distribution changes among the versions in chronological defect datasets. Findings from streaming data analytics verify that if the historical data change over time, the prediction models become outdated [4]. Likewise, Ekanayake et al. [14,15] confirmed that the accuracy of SDP models fluctuates if the chronological data change over time, meaning that the prediction performance varies. In order to construct prediction models that are more efficient, the research community should address the challenge posed by evolving data distributions over time, as elucidated by Turhan [7]. Therefore, it is necessary to be conscious of dataset shifts and their implications in the SE domain. Dataset shifts can generate biased prediction results, for example, due to changes in class prior probabilities. Alterations in data probabilities, encompassing shifts in probability distributions, can result in erroneous outcomes. Even a prediction model that is well trained can become obsolete in the presence of such drift, as highlighted by Dong et al. [8]. Furthermore, other researchers [9–13] have reported that the distribution changes among the versions in chronological defect datasets. Empirical evidence from the field of streaming data analytics corroborates that, as temporal shifts occur in historical data, prediction models experience obsolescence [4]. Likewise, Ekanayake et al. [14,15] confirmed that the accuracy of SDP models fluctuates if the chronological data change over time, meaning that the prediction performance varies.

1.1. Research Questions

In this paper, we describe our systematic investigation to detect CD in chronological defect datasets (i.e., in the scenario of CVDP) to answer the following research questions.

RQ1: Is there any data distribution difference across software versions that degrades the prediction performance?

RQ1 is characterized by the null hypothesis H0: There is no data distribution difference across project versions.

RQ1 is answered by proposing and evaluating a framework that we term concept drift detection (CDD). The framework includes a detection method to identify CD across versions by providing information about when to update or re-train the prediction models. The adopted statistical hypothesis test—a chi-square test with Yates's continuity correction [16]—is designed to identify the statistically significant differences across project versions, and the practical significance is computed by adopting a robust effect-size computational method, namely Cliff's delta (δ).

RQ2: Can insights be gained by observing the trends in CD detection results while the window size varies?

RQ2 is observationally addressed by analyzing the trends of experimental results across the versions and by varying the window size covering the training set over time to avoid the ill effects of CD. Specifically, we use the following:

1. All data from past versions, excluding the most recent one, with a window size of $n = N$, where N encompasses all historical data;
2. Data from the two most recent completed versions prior to the latest completed version, where the window size $n = 2$;
3. Data from the most recent completed version prior to the latest completed version, where the window size $n = 1$;
4. As a control, data from the most recent completed version are used as a testing set to obtain prediction results.

1.2. Contributions

The main contributions of the work are listed as follows:

1. This study is one of the few initial contributions to conduct theoretical analysis and identification of CD in software defect datasets.
2. A CDD framework is proposed to recognize CD in cross-version defect datasets.
3. The potential effects of CD on CVDP performance are empirically evaluated.

We encourage the use of the CDD framework for the identification of CD in chronological defect datasets for CVDP. Having the capacity to detect CD, software quality teams will have the opportunity to implement the required measures for enhancing the prediction models related to CVDP.

The subsequent sections of this paper are structured as follows: in Section 2, we delve into the elucidation of key concepts, namely dataset shift, drift detection methods, and concept drift. Section 3 offers an intricate description of our framework, providing a step-by-step breakdown of its components. This section is dedicated to outlining the experimental settings used to assess the effectiveness of our proposed framework. Our experimental results are detailed in Section 4, while Section 5 is dedicated to the analysis and discussion of these results. In Section 6, potential threats to the study's validity are identified and discussed. Section 7 explores relevant prior research, specifically focusing on studies pertaining to cross-version defect prediction (CVDP). Lastly, we draw the paper to a conclusion in Section 8, summarizing the key findings and providing our concluding remarks.

2. Preliminaries

2.1. The Notion of Dataset Shift

A vast amount of data are now available in streaming format as a result of the increased pervasiveness of digitization. Streaming data are typically thought to be non-stationary in nature [17]. The data-generating process in non-stationary environments is variously characterized as data shift, or concept drift [17]. Multiple researchers in the field of data-stream mining have offered definitions for this phenomenon, which is also referred to as "prior probability shift" [4,7,18]. Data drift is observed when there is a change in the probability of the occurrence of event X at a specific time, denoted as time t. It can be described formally as:

$$P_t(X) \neq P_{t+1}(X) \tag{1}$$

Data drift can be understood by considering two distinct data distributions referred to as historical data and new data, which correspond to different timeframes or perspectives. In Figure 1, the historical data D_{t+1} to D_{t+6} are regarded as the new data and the old window D_{t+7} are considered as the recent window, and are determined by the individual as indicated by Lu et al. [4]. As new data accumulate over time, the windows move continuously.

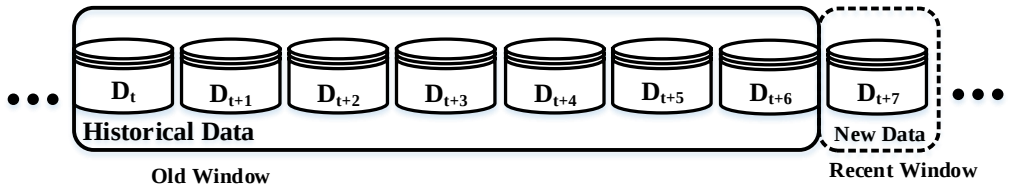

Figure 1. Data distribution using two windows at a timestamp $t + 7$.

To quantify the severity of drifting data, Figure 2 portrays the severity of a scenario that starts at the distribution D_{t+3} and ends at D_{t+7}. The start and end points have a decisive effect on the learning process, which in turn determines the effectiveness of the prediction model. The prediction model often requires an update in order to adjust for variations in data distribution, as is well recognized in the literature [4,8,19–21].

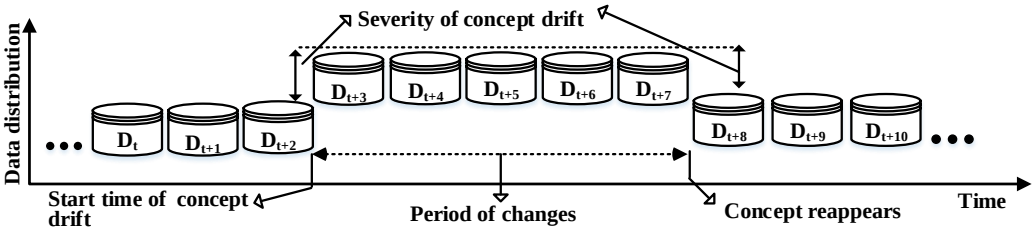

Figure 2. Severity of data drift in occurrence time. The changes occur at time $t + 2$ and $t + 7$.

2.2. Approaches of Detecting Data Drift

The method that is most widely used in the literature on data streaming is that of change detection in the underlying distribution. Change-detection methods usually monitor the error estimates of prediction models. The drift detection method (DDM) [22] and early drift detection method (EDDM) [22] are the most widely used change-detection methods in data-stream mining research.

DDM is a learning framework that is considered applicable to any learning model [22]. When new data are available, the learning model makes a judgment. If the sample i in the distribution is stationary, the error-rate p_i of the learning model will decrease. However, if the distribution changes over time, p_i will increase and the actual model will gradually become inappropriate. The drift detection method (DDM) was the pioneering model to introduce the concept of defining warning and drift levels based on alterations in the data distribution. The warning level is reached if:

$$p_i + s_i \geq p_{min} + 2s_{min}, \tag{2}$$

where the standard deviation is:

$$s_i = \sqrt{(p_i(1 - p_i)/i)} \tag{3}$$

The drift level is reached if:

$$p_i + s_i \geq p_{min} + 3s_{min} \tag{4}$$

The developers of DDM set the threshold parameters to 0.95 and 0.90 for the warning level and drift level, respectively.

In contrast, EDDM takes into account the average separation of two error rates p_{i} and the standard deviation s_{i} of the classification model. This approach stores the values of p_{i}, s_{i} and the maximum values of p_{max} and s_{max}. When $(p_{i} + 2s_{i})/(p_{i} + 2s_{i}) < \alpha$, the warning level for drift detection is triggered. When $(p_{i} + 2s_{i})/(p_{i} + 2s_{i}) < \beta$, drift has actually occurred. The developers of EDDM set the values of α and β to 0.95 and 0.90. To detect drift in streaming data, both DDM and EDDM make use of a single data instance and rely on predefined thresholds.

2.3. Concept Drift

The concept of dataset shift or drift was first identified in the data-stream mining community [7,23,24]. Detecting the dataset drift is still very challenging in non-stationary environments, as in these environments, the data change over time [25]. Lu et al. [23] pointed out that the uncertainty of a dataset is an inherent property of streaming data. Change is unavoidable and inevitably deteriorates the prediction results. The concept of dataset shift in software engineering-prediction models was introduced by Turhan [7], who cautioned about its effects and attempted to account for changes in data distributions. However, drift-identification techniques used for the analysis of chronological software datasets remain suboptimal. As shown by [15,26,27], the accuracy of such prediction models degrades over time due to the changes in software-defect datasets.

The concept of dataset shift refers to the changes in data distributions [28]. More formally, the problem is formulated as follows. Suppose that t is the discrete time index, and v is the version index of a chronological dataset.

$$\Xi = [v_1, v_2, v_3, \ldots, v_N] \tag{5}$$

$$\Phi = \begin{bmatrix} m_{11} & m_{21} & \ldots & m_{N1} \\ m_{12} & m_{22} & \ldots & m_{N2} \\ . & . & \ldots & . \\ . & . & \ldots & . \\ m_{1p} & m_{2p} & \ldots & m_{Np} \end{bmatrix} \tag{6}$$

The total dataset consists of a number of N in versions Ξ of the dataset. By Equations (5) and (6), we can define the dataset as $\{\Xi, \Phi\}$ where each version v contains the number of p in modules $m = 1, 2, 3, \ldots, p$. In Equation (6), matrix Φ is constructed to describe the modules m of the versions Ξ. As an example, suppose there are two versions, v_1 and v_2, at discrete time t. The data of these two distributions change over time. If the joint probability changes in the data of versions Ξ, it can be defined as $v_i \neq v_j$ where $(i, j) = 1, 2, 3, \ldots, p$.

3. Materials and Methods

Within this section, we present an intricate overview of our framework and the meticulous details of our experimental configuration. This encompasses a comprehensive introduction to the chronological defect datasets we have employed and an in-depth, step-by-step illustration of the CD detection process, utilizing the sophisticated chronological split approach.

3.1. Benchmark Datasets

In this empirical research endeavor, we explored a set of 30 distinct versions originating from seven benchmark software projects, all of which have been made available through the dataset curated by Jureczko and Madeyski [29]. These chronological datasets have seen extensive use in numerous studies (i.e., [9,30,31]) of CVDP. These datasets are gathered from the SEACRAFT (https://zenodo.org/communities/seacraft (accessed on 10 July 2023)) repository of empirical software engineering data. More detailed descriptions of the datasets can be found in [32,33].

Table 1 displays the benchmark datasets' statistics where #M, #FM, and %FM describe the number of modules, the number of faulty modules, and the percentages of faulty modules, respectively. Note that the current versions inherit many modules from the prior versions. Some modules are added or deleted from the previous between versions, which leads to making a difference across between theme versions. Table 1 also shows the values of skewness and kurtosis of the datasets. If the value of skewness is higher or greater than $+1$, it indicates a skewed distribution. If the value of kurtosis is greater than $+1$, it seems to indicate that the distribution is peaked [34]. The dataset consists of modules, with each module being categorized as either defective or non-defective. Additionally, these datasets

incorporate 20 static code metrics for analysis (Table 2) for the prop projects' datasets and 21 static code metrics (Table 3) prepared by the NASA metrics data program [35] for the JM defect datasets.

Table 1. Statistics of the benchmark dataset.

Project	Version	# M	#FM	%FM	Skew	Kurt
jm	1	7782	1672	21.49%	1.39	2.93
	1.2	9593	1759	18.34%	1.64	3.68
	1.3	7782	1672	21.49%	1.39	2.93
prop-1	9	4255	149	3.50%	5.06	26.59
	44	4620	389	8.42%	2.99	9.97
	92	3557	1269	35.68%	0.60	1.36
	128	3527	220	6.24%	3.62	14.10
	164	3457	319	9.23%	2.82	8.94
	192	3598	85	2.36%	6.27	40.35
prop-2	225	1810	147	8.12%	3.07	10.40
	236	2231	76	3.41%	5.14	27.39
	245	1962	103	5.25%	4.01	17.10
	256	1964	625	31.82%	0.78	1.61
	265	2307	229	9.93%	2.68	8.18
prop-3	285	1694	177	10.45%	2.59	7.69
	292	2285	209	9.15%	2.83	9.03
	305	2344	89	3.80%	4.83	24.38
	318	2395	365	15.24%	1.93	4.74
prop-4	347	2871	162	5.64%	3.84	15.78
	355	2791	924	33.11%	0.72	1.52
	362	2854	213	7.46%	3.24	11.48
prop-5	4	3022	264	8.74%	2.92	9.54
	40	4053	466	11.50%	2.41	6.83
	85	3077	948	30.81%	0.83	1.69
	121	2998	425	14.18%	2.05	5.22
	157	2496	367	14.70%	1.99	4.97
	185	2825	268	9.49%	2.77	8.65
prop-42	452	256	33	12.89%	2.21	5.91
	453	192	20	10.42%	2.59	7.72
	454	212	13	6.13%	3.66	14.37

Table 2. Description of the static metrics for prop dataset.

Metric	Description
WMC	Each class using weighted methods
DIT	Tree depth of the inheritance
NOC	Children in the sample
CBO	Connecting classes of objects
RFC	Class's response
LCOM	Methods are not cohesive enough
Ca	Afferent couple
Ce	Successful coupling
NPM	The quantity of public methods
LCOM3	Methods are not cohesive and from LCOM
LOC	Lines of code
DAM	Access to data metrics
MOA	Aggregation measurement
MFA	Functional abstraction index
CAM	Coherence of classification methods

Table 2. *Cont.*

Metric	Description
IC	Coupling with inheritance
CBM	Linking different methods
AMC	Method complexity on the average
MaxCC	The highest possible values for a class's methods' cyclomatic complexity
Avg(CC)	Calculating the average cyclomatic complexity of a class of methods
BUG	Bugs in the class

Table 3. Description of the metrics for JM dataset.

Attribute	Description
V(g)	Cyclomatic complexity
Iv(G)	Design_complexity
Ev(G)	Essential_complexity
loc	Loc-total
n	Halstead total operators + operands
v	Halstead total operators + operands
l	Halstead "volume"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead
t	Halstead's time estimator
lOCode	Halstead's line count
lOComment	Halstead's count of lines of comments
lOBlank	Halstead's count of blank lines
lOCodeAndComment	
uniq_Op :	unique operators
uniq_Opnd :	unique operands
total_Op :	total operators
total_Opnd :	total operands
branchCount :	of the flow graph
defects {false,true}:	module has/has not reported defects

3.2. Framework

This framework contains three primary stages: (1) data streams; (2) concept drift detection and understanding; and (3) concept drift adaptation. Figure 3 describes the details.

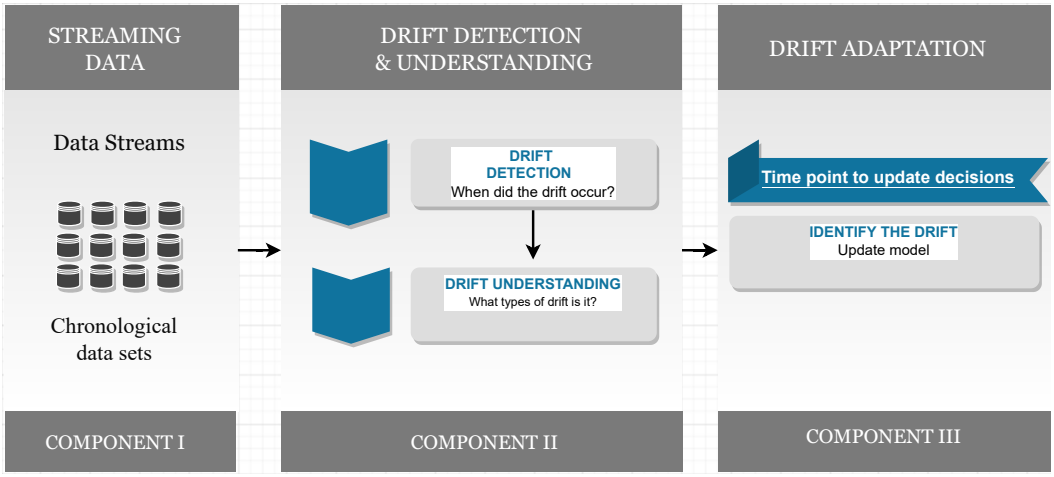

Figure 3. CDD, a CD detection and adaptation framework.

Stage I collects all available data from the sources and restructures them in a time frame. The data can be reformed according to the user demands as instructed by Lu et al. [23]. For example, if the data come in chronological order, therefore, we could use a chronological split approach to process the data and make the scenario as chronological splitting for CVDP. We can also refer this step as data preprocessing. This stage is considered an important part for building a practical scenario.

Stage II identifies the CD if the data change over time. For recognizing CD in the considered data streams, it triggers three flags; CD, warning, and control. The CD flag reports the changes in the compared distributions that deteriorate the prediction performance. The warning flag is triggered when it tends to be CD, whereas the control flag refers to the stationary distributions. This triggering process helps to know the state of CD and where the CD is located.

CD adaptation is dependent on the knowledge gained from stage II. In the CD adaption stage (stage III), the model is updated or retrained according to the result of CD detection. Section 3.3 describes this framework step by step, as implemented by chronological defect datasets and applied for CVDP. In this paper, we conduct the experiment by focusing the detection of CD from where the guidance can be taken on when to update or retrain the prediction model.

3.3. CDD Implementation

Stage I: Data streams and CVDP scenario design

The software project experiences several releases because of adding and deleting the software functions [9,31]. The CVDP scenario is designed in such a way where the software project contains several versions and behaves as streaming data. Since the changes occur in every version of the software projects that release at different times, the motivation to carry out this experiment is to further examine whether the changes in the data of each version would lead to CD causing poor prediction results.

Each version of the application system is referred to as windows when describing stage I. As an example, two consecutive versions can be deliberated as two windows—where one window is considered as training, and the other one is the test set in a window-based approach. Motivated by the studies of software cost estimation [36,37], we assume that each version arrives one-by-one, taking a chronological split approach into account for CVDP. Similarly, the splitting fashion also exhibits a window-based approach called a data management technique, as described by research in the field of data stream mining [38].

The chronological split approach is assessed in our CD detection framework: it divides the accessed data and considers training set in three ways: (1) the entire history of past versions as window-of-size-n where n refers to all historical data (i.e., called a full-memory approach in data streaming mining literature [38]) except the recent version of the software project (Figure 4); (2) one recent version's data prior to the recently completed version as window-of-size-1 (i.e., called a no-memory approach [38]) (Figure 5); and (3) completed recent two version's data as window-of-size-2 approach (Figure 6). Furthermore, the window-of-size-1 and window-of-size-2 approaches are the forgetting mechanism of the modules of the non-stationary environment. We employ two-window-based approaches (see Figure 1) considered to be the training set and the currently completed version considered to be test set.

In the learning process, the most recently completed version is considered to be the testing set to assess the model. In CVDP, the potential shortcoming is that of how many data need to be considered for learning as data arrive version by version and behave as streaming data. The above-mentioned chronological splitting approach is more closely related to practice. Because of this, the most practical issue is to form the window as suggested by Iwashita et al. [39].

Describing the chronological splitting approach for CVDP, the window-of-size-n (full-memory approach) employs all the historical data, which means not forgetting the old versions of the software project. In window-of-size-1 (no-memory approach), the learner

induces only one completed version's data that is prior to the most recent completed version (Figure 5). For example, version 1 was used as a train set to build the model for the jm project dataset. After that, the model was tested on the instances of version 1.2. In window-of-size-2, the learner only induces two completed version's data that is prior to the most recent completed version (Figure 6).

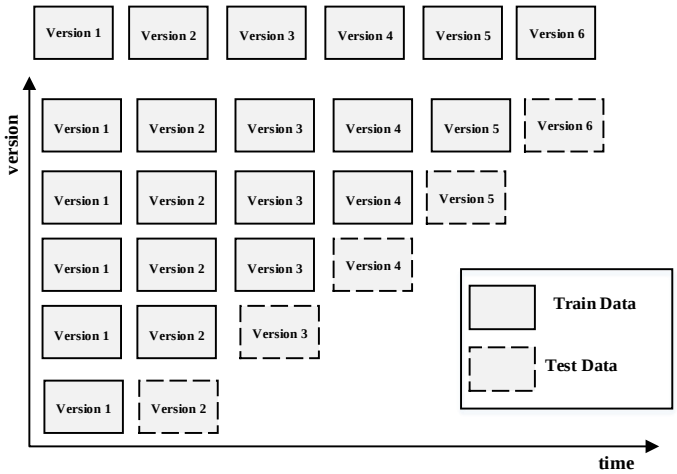

Figure 4. Window-of-size-n where *n* refers to all available historical data.

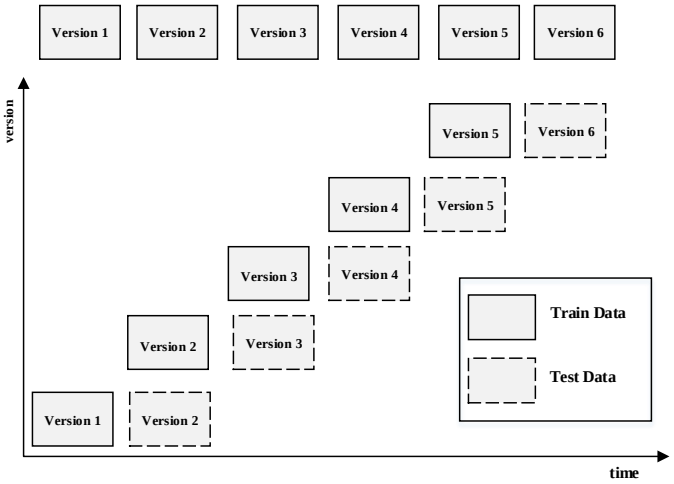

Figure 5. Window-of-size-1.

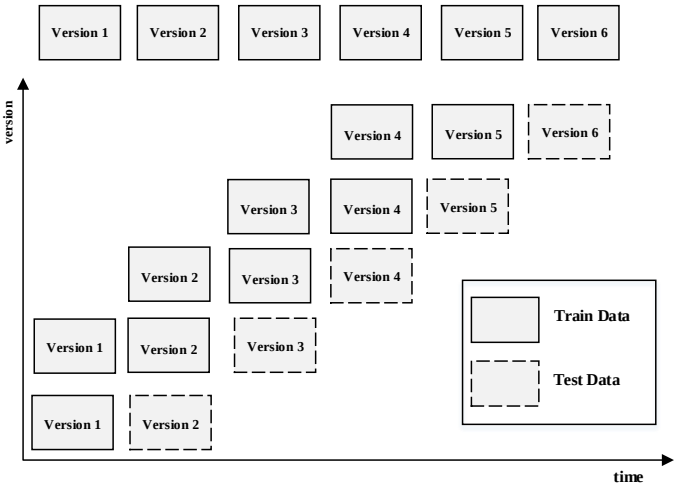

Figure 6. Window-of-size-2.

Stage II: CD detection and understanding

For detecting CD, we operate under the assumption that data arrive in a streaming fashion, and the surrounding environment undergoes non-stationary changes. In this environment, it is difficult to assume that the instances are identically distributed and the distributions change over time.

In this CD detection process, it is assumed that the data arrive in batches of instances at a time. Supposedly, the instances can be presented as a pair of $(\overrightarrow{a}, b)$ where $\overrightarrow{a}$ are the values with different attributes and b is the label of class. For the $i-th$ instances, it can be presented as $(\overrightarrow{a_i}, b_i)$. When the prediction model is built, the class level can be either $(b_i = \acute{b_i})$ or $(b_i \neq \acute{b_i})$ where the error-rate is a random variable in Bernoulli trails. The general scheme is adopted from the drift detection method, DDM [22]. In the binomial distribution, the probability of the random variable illustrates the errors of i instances. For each point i, the error-rate er_i is computed with its corresponding standard deviation:

$$sd_i = \sqrt{(er_i(1-er_i)/i)} \tag{7}$$

According to probably approximately correct (PAC) learning theory [40], the distribution is stationary if the error-rate of the learning model decreases. Changes in the error-rate of learning models increase the probability of the distribution being non-stationary. Therefore, the error-rate er_i of the learning model will decrease if the number of samples i in the distribution is stationary. The distribution changes over time if the error-rate er_i of the learning model increases and the actual model becomes inappropriate. The binomial distribution is considered a normal distribution for a large number of data distributions, and consequently, it contains approximately the same variance and mean. The probability distribution should not be changed unless the changes in the data distributions happen and then the $1-\frac{\alpha}{2}$ confidence interval for error-rate er_i with i instances is $(er_i \pm \alpha * sd_i)$ where the parameter α relies on the confidence level.

For managing the lowest possible values of the error rate and average deviation throughout the learning process, two registers are assigned, er_{min} and sd_{min}, accordingly. When the new instances become available, the error-rate and its standard deviation are computed during the modeling process. It is compared with the existing er_{min} and sd_{min}. If it becomes lower than the assigned values of registers, the registers are kept updated with the new minimum values. In cross-version settings, as an example, every version of chronological defect datasets contains a sufficient number of data distributions. Each time the data of new versions become available, the instances i are processed to learn the model to obtain its corresponding error-rate er_i and standard deviation sd_i and compared with stored register values to keep updated. For detecting CD in cross-version settings, we set the confidence level for triggering the warning flag to 95% $(\alpha = 0.05)$ with the threshold $er_i + sd_i \geq er_{min} + 2sd_{min}$. The confidence level for triggering the CD flag was determined to 99% with the threshold $er_i + sd_i \geq er_{min} + 3sd_{min}$ as suggested by Gama et al. [22].

For detecting CD in the data distributions, we define three flags: warning K_w flag, CD K_{CD} flag, and control K_c flag. We discuss the triggering process as follows.

1. Warning flag K_w: The learning process maintains the current decision model and it triggers the warning flag if it reaches the threshold $er_i + sd_i \geq er_{min} + 2sd_{min}$.
2. CD flag K_{CD}: It triggers the CD flag if it reaches the threshold $er_i + sd_i \geq er_{min} + 3sd_{min}$.
3. Control flag K_c: When the learning error does not reach the level of K_w and K_{CD}, it is assumed that the environment is stationary.

For replicating this study, we provide the pseudo-code for the CD detection process described in Algorithm 1. The algorithm employs the base classifier on the train set and generates the error-rate of the prediction model (lines 1 and 2). We determine the train window T_r and test window T_s according to the considered chronological splitting approaches (as can be seen in Figures 4–6). After calculating the standard deviation and

finding the minimum values of error-rate er_{min} with standard deviation sd_{min} and stored in register, we identify the CD, warning, and control flags (lines 8–13).

Algorithm 1 Overview of *CDD*

Input:
Training window T_r and testing window T_s
Learner $BaseClassifier$
Sample of training and testing i
Outcome:
CD flag K_{CD}, warning flag K_w and control flag K_c.
Procedure:

1: **for** $t = 1, 2, \ldots$ **do**
2: Learner construction on T_r
3: Prediction of the of the learner on T_s
4: Rate of error calculation er_i
5: Compute $sd_i = \sqrt{(er_i(1 - er_i)/i)}$
6: Preserve er_i, sd_i and discover the values of register as lowest value of er_{min} with sd_{min} during the process of learning
7: **if** $(er_i + sd_i \geq er_{min} + 3sd_{min})$ **then**
8: K_{CD} detected
9: **else if** $(er_i + sd_i \geq er_{min} + 2sd_{min})$ **then**
10: K_w detected
11: **else**
12: K_c
13: **end if**
14: **end for**

The CD detection process helps to include a discussion of when to enhance the prediction models in the learning process over time. By comprising the triggering process, the three types of flags attained help to make a conclusion sharper with regard to when to enhance or update the models. CD flags show that the CD is present when immediate action is required to produce more accurate predictions. The target distributions are generally CD, according to warning flags. If not, the control flag is triggered, meaning that target data distributions are stationary, which prevents changes over time.

Stage III: CD adaptation

In the situation where there is uncertainty in the streaming data, the learning model is often challenged to adapt the changes. It is a challenging task to address the changes in such distributions, which change over time [38]. The most common remedy in such situation is to update or enhance the existing model, as recommended by Lu et al. [41]. By knowing where the CD occurs and how it happens in the distributions, the learning model could be enhanced and personalized.

3.4. Designing CD Detection Statistical Test

We adopt a statistical test and design it for a two-window-based data distribution in cross-version settings for achieving statistically significant CDs. We assume that the two distributions indicate two subsequent versions. In addition, a chronological splitting approach is adopted to maintain the order of the software's versions. Moreover, it is also referred to as two-window-based data distribution. In the CVDP scenario, two contiguous software's versions exhibit two-window-based data distributions.

The implementation of a statistical method primarily serves to make sure that the CD is not a result of an error in sampling and that the CD results are statistically noteworthy. The implementation of the chi-square test with Yates's continuity correction is applied to window-based data distributions utilizing 2×2 contingency tables displayed in Tables 4 and 5 below.

Table 4. Recent window—confusion matrix.

	Predicted as Positive	**Predicted as Negative**
Actual positive	q_1	r_1
Actual negative	s_1	t_1

Table 5. Old window—confusion matrix.

	Positive Prediction	**Negative Prediction**
Actual positive	q_2	r_2
Actual negative	s_2	t_2

According to Tables 4 and 5, the adapted contingency table (i.e., for recent and old windows) is displayed in Table 6 which is defined below.

Table 6. Adapted contingency table—recent and old window.

	Recent Window R_i	**Old Window** O_i
# of correct class C_c	$q_1 + t_1$	$q_2 + t_2$
# of errors O_t	$r_1 + s_1$	$r_2 + s_2$

Let R_i and O_i be the labeled two-window-based chronological datasets that have two feature spaces transformed into $V_i = q_1, r_1, s_1, t_1$ and $V_j = q_2, r_2, s_2, t_2$ of the recent R_i and old O_i window where q_1, t_1 are the correctly classified and s_1, r_1 are the misclassification among the actual positive and negative values. Similarly, for the old window (in Table 5), q_2, t_2 are correctly classified and s_2, r_2 are incorrectly classified among the positive and negative values, accordingly. Tables 4 and 5 provide descriptions of the confusion matrix for the current and previous windows, respectively.

According to the 2×2 contingency table, the probability of instances wherein recent and old windows are correctly and incorrectly classified as one or the other is presented in Table 6 and determined by Equation (8) where $q_1 + t_1$ denotes the number of correct class R_iC_c among the total instances $(q_1 + t_1 + r_1 + s_1)$ of the recent window R_i. $r_1 + s_1$ denotes the number of incorrect class R_iO_t among the total number of instances $(q_1 + t_1 + r_1 + s_1)$ of recent window R_i. Again, $q_2 + t_2$ denotes the number of correct class O_iC_c among the total instances $(q_2 + t_2 + r_2 + s_2)$ of the old window O_i. $r_2 + s_2$ are the number of incorrect class O_iO_t among the total instances $(q_2 + t_2 + r_2 + s_2)$ of the old window O_i. The statistic derived from Equation (8) is analogous to the chi-squared test with Yates's continuity correction. This statistical approach helps prevent the overestimation of significant values, particularly for small data distributions, and is a method commonly employed in the literature, as evidenced by references such as [42,43].

$$T(O_iC_c, O_iO_t, R_iC_c, R_iO_t) = \frac{\mid \frac{O_iC_c}{O_iO_t} - \frac{R_iC_c}{R_iO_t} \mid -0.5(\frac{1}{O_iO_t} + \frac{1}{R_iO_t})}{\sqrt{\frac{R_iC_c+O_iC_c}{R_iO_t+O_iO_t}(1 - \frac{R_iC_c+O_iC_c}{R_iO_t+O_iO_t})(\frac{1}{R_iO_t} + \frac{1}{O_iO_t})}} \quad (8)$$

The two sequential data distributions in the temporal versions of defect datasets are not identical if and only if the calculated p-values are below the threshold for significance. Considering that the measured values of p (0.05) are lower than at the 5 percent significance level, the null hypothesis is that the distribution on both windows is not identical.

3.5. Evaluation Settings

To evaluate the ability of the CDD framework to detect CD in the chronological defect datasets, we adopt naïve Bayes (NB) as a single base learner to classify the modules and their output, with the latter to be used in class prediction. We use NB because it is simple and widely used in data-stream mining. We evaluate the models using the confusion matrix (see Table 7) and in terms of their error rate, which is computed as:

$$error - rate = 1 - accuracy \tag{9}$$

We compute accuracy as:

$$accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{10}$$

where the outcomes are categorized in one of four ways: true positives (TP) represent the instances where the model correctly identifies defective modules as such. False positives (FP) occur when the model incorrectly categorizes non-defective modules as defective. True negatives (TN) denote situations where the model accurately identifies non-defective modules as non-defective, and false negatives (FN) refer to cases wherein the model erroneously labels defective modules as non-defective.

RQ1: Is there any data distribution difference across software versions that degrades the prediction performance?

While assessing the prediction performance, the objective of the models is to identify the defective modules from minority classes. Menzies et al. [44] argued against using only overall accuracy for performance assessment because of the unequal error-cost associated with minority classes. Other researchers [45,46] used the probability of a false alarm (pf) as recommended by Menzies et al. [44] as a stable evaluation metric to assess the performance of classification models. However, the key objective of this empirical study is to identify the CD across versions, where the error-rates of the models were analyzed based on our CDD framework. From Table 7, it can be seen that the mathematical definition of pf used for this experiment is:

$$pf = \frac{FP}{(FP + TN)} \tag{11}$$

Low values of pf correspond to a better value for prediction performance [47].

Table 7. Confusion matrix.

	Predicted Positive	**Predictive Negative**
Actual positive	TP	FN
Actual negative	FP	TN

Regarding the chronological splitting of the versions, we utilize an approach called a "two-window-based approach" in our proposed CDD framework. As described earlier, we consider three versions of the splitting procedure: a window-of-size-n (Figure 4); a window-of-size-1 (Figure 5); and a window-of-size-2 (Figure 6)—bearing in mind that consecutive versions share similar characteristics, which helps to train the model accurately, as suggested by Amasaki [48].

RQ2: Can insights be gained by observing the trends in CD detection results while the window size varies?

In Section 3.3, the procedures of chronological splitting are outlined in stage I. In the three aforementioned splitting approaches, the training and test sets change in a streaming

manner. For a better understanding of the learning process in the chronological splitting (when the data of the new version become available), the recent window is updated (referred to as the moving window). In the approach of a window-of-size-n, all old versions are summed into a single package containing all of the completed versions. In doing so, the size of the window is increased. This procedure was also adopted by Lokan and Mendes [49] in a software-effort estimation studies. As an example, for the project prop-1, there are six versions available. In the window-of-size-n splitting approach, the first two versions (i.e., 9 and 44) are considered to be the training set (old window) and the next version, 92, is considered to be the testing set (recent window). The prediction model is built and results are computed. When version 128 becomes available, the data of version 92 along with the previous two versions are considered to be the training set (old window) and version 128 is considered to be the testing set (recent window). This procedure is iterated until all of the versions of the datasets of interest have been taken into consideration.

The error-rate is computed each time the new version's data become available for the mentioned splitting processes (see stage I in Section 3.3). Note that the ability to identify the best among all of the prediction models is out of the scope of this study.

Based on the CDD procedure, the CD detector triggers one of three flag values (warning, CD, and control) to indicate the CD status in the target data distributions, as discussed in stage II (see in Section 3.3). The statistical significance test serves the purpose of confirming that the CD values are indeed statistically significant, thus helping to rule out the possibility that these CDs are merely a result of sampling error.

3.6. Cliff's δ Effect Size Computation

Numerous studies in disciplines like behavioral science have used the quantitative measure of an impact's magnitude known as "effect size" (ES) [50]. According to Carl et al. [51], it is advisable to employ effect sizes (ESs) in empirical studies as they can contribute to obtaining results that are not only statistically significant but also practical and meaningful.

Moreover, to strengthen our empirical technique, we perform a comparison of the chronological split approach with Cliff's delta, a widely adopted and robust effect-size computation method, (Cliff's δ), [52] to quantify the practical significance of differences across the defect datasets.

RQ1 characterizes the hypothesis tested in this study. To avoid the inaccurate results of hypothesis testing, it is recommended by Kampenes et al. [53] to acknowledge the effect sizes in the experimental evaluation of software engineering (SE) and to use a statistical significance test to obtain meaningful outcomes. In this context, a 1% increase in the predicted defect results would be likely to have a highly negative effect if the defects are critical.

Kitchenham et al. [54] suggested the utilization of Cliff's delta (δ) as an effective method for calculating effect sizes due to its capacity to manage circumstances where the results are deadlocked. This method yields an effective measure of two-window-based data distributions having different distribution sizes, which indicates the practical usefulness of an empirical SE study [10]. In our analysis, the interpretation of practical significance and the calculation of effect sizes draw upon the magnitude thresholds outlined by Kampenes et al. [53]. These thresholds provide a structured framework for understanding the significance of the observed effects. A negligible effect, for instance, is indicated when the effect size (δ) is less than 0.112, signifying a minimal practical impact. A small effect is characterized by an effect size falling in the range of 0.112 to less than 0.276, suggesting a modest yet noteworthy influence. When the effect size ranges from 0.276 to less than 0.427, it is deemed a medium effect, indicating a more substantial impact. Finally, a large effect is inferred when the effect size is equal to or greater than 0.427, highlighting a substantial and practically meaningful consequence. These thresholds aid in providing a clear understanding of the magnitude and relevance of the observed results in our empirical study. As recommended by [54], we implement Cliff's δ method using the effsize R package.

4. Results

In this section, we present the results of the empirical study to identify the CD of the considered defect datasets and evaluate the models' performance using their error-rate, pf, and a base learner, NB. The experimental results are presented based on the CDD framework by taking chronological splitting into account and performance accessed using Cliff's δ effect size, chi-squared with Yates's continuity correction, and the Nemenyi test at 5% significant level.

Table 8 presents the overall prediction performance of CDD across the versions of our considered chronological datasets. Within this table, the terms "IVersion," "#InsOWin," and "#InsRWin" correspond to specific attributes. "IVersion" stands for the initial version, while "#InsOWin" and "#InsRWin" represent the counts of instances within the old window and recent windows, respectively. Column five (from the left) shows the CD values, in which "*" is used to denote the versions where CDs are identified. The rightmost column shows the statistically significant values calculated by the adopted statistical test, denoted by "**" in this Table 8, where the statistically significant ($p < 0.05$) differences between the data distributions associated with the projects' versions are identified. This identification helps the user decide when to re-train or update the prediction models. Stage III of our proposed CDD framework uses this information in a process called CD adaptation, which indicates the ideal time-point at which to take appropriate action so that the prediction model is made or kept reliable.

In addition, we found that the software projects' versions for which the differences of data distributions are practically significant exhibited large effect sizes, as illustrated in Table 9. It is worth mentioning that the versions identified by CDD as having a statistically significant CD also exhibit large-effect sizes in the pairwise comparison, except for the data of version 355 of the project prop-4. Regarding the question of which splitting approach performs best, it can be seen that the moving-window-based approach, specifically window-of-size-1, achieves the largest effect size for CD detection (i.e., $\delta \geq 0.427$).

Figure 7 portrays the CD values associated with the versions of the software projects for which the differences in the data distributions are statistically significant. The significant values of CDs are marked by the vertical lines, which indicate that, at this point, the user should make a decision to update or re-train the model. According to our proposed CDD framework, these are the optimal time-points to make these decisions to ensure the reliability of the prediction model.

By closely observing the CD values (see Table 8), it can be seen that the CD detection procedure of our proposed CDD framework identifies statistically significant CDs in the considered chronological datasets. It can also be seen that the CDs identified in the data distribution are generally identical among the three adopted chronological split approaches. One exception is observed for the prop-1 dataset, which contains six versions. Window-of-size-2 identifies the most statistically significant CDs of this dataset, in that CDs are identified for the first four versions (i.e., 91, 44, 92, and 128) by the CD detector. Among these versions, the CDs are statistically significant in versions 44, 92, and 128, again the maximum number found by any of the splitting approaches, meaning that the window-of-size-2 performs better only for the prop-1 dataset. The experimental results reveal that the exponentially increasing error-rate across different software versions is associated with CD, which consequently leads to poor prediction performance.

Table 8. Results obtained from the adopted the three chronological split approaches by the CDD technique and the chi-squared test with Yates's continuity correction for finding significant CDs' from the chronological defect datasets for CVDP. Here, "*" is used to denote the versions where CDs are identified, "**" means significant values, the CDs are identified and marked in the table by changing the background color of the table's cells.

Window-of-Size-1						
Project	**IVersion**	**#InsOWin**	**#InsRWin**	**CD**	**t-Value**	***p*-Value**
prop-1	9	4255	4620	8.153 *	5.422	0.116
	44	4620	3557	91.621 *	24.591	**0.026** **
	92	3557	3527	89.879 *	23.172	**0.027** **
	128	3527	3457	8.310 *	2.173	0.275
	164	3457	3598	0.998	6.331	0.100
prop-2	225	1810	2231	5.260 *	3.661	0.170
	236	2231	1962	0.995	0.085	0.946
	245	1962	1964	30.591 *	12.821	**0.050** **
	256	1964	2307	33.783 *	9.171	0.069
prop-3	285	1694	2285	5.973 *	0.626	0.644
	292	2285	2344	0.999	1.553	0.364
	305	2344	2395	16.202 *	5.765	0.109
prop-4	347	2871	2791	26.242 *	21.006	**0.030** **
	355	2791	2854	1.003	17.890	0.036
prop-5	4	3022	4053	0.991	2.366	0.255
	40	4053	3077	28.760 *	13.823	**0.046** **
	85	3077	2998	11.260 *	10.779	0.059
	121	2998	2496	4.637 *	0.856	0.549
	157	2496	2825	2.632	3.433	0.180
prop-42	452	256	192	2.280	2.562	0.237
	453	192	212	3.000 *	9.811	**0.045** **
jm	1	7782	9593	11.689 *	19.296	**0.033** **
	1.2	9593	7782	9.422 *	9.811	0.065
Window-of-Size-N						
Project	**IVersion**	**#InsOWin**	**#InsRWin**	**CD**	**t-Value**	***p*-Value**
prop-1	9	4255	4620	4.887 *	5.422	0.116
	44	8875	3557	48.356 *	32.327	**0.020***
	92	12,432	3527	3.994 *	10.491	0.060
	128	15,959	3457	6.396 *	6.070	0.104
	164	19,416	3598	1.069	12.759	0.050
prop-2	225	1810	2231	7.216 *	3.661	0.170
	236	4010	1962	1.006	2.307	0.260
	245	6003	1964	34.406 *	15.811	**0.040** **
	256	7967	2307	6.282 *	3.787	0.164
prop-3	285	1694	2285	4.693 *	0.626	0.644
	292	3973	2344	0.933	2.208	0.271
	305	6323	2395	11.838 *	7.441	0.085
prop-4	347	2871	2791	39.092 *	21.006	**0.030** **
	355	5662	2854	0.953	11.578	0.055
prop-5	4	3022	4053	2.627	2.366	0.255
	40	7075	3077	30.946 *	16.657	**0.038** **
	85	10,152	2998	10.129 *	2.901	0.211
	121	13,150	2496	5.532 *	2.969	0.207
	157	15,646	2825	1.210	7.181	0.088
prop-42	452	256	192	1.148	2.562	0.237
	453	448	212	4.319*	14.507	**0.044** **
jm	1	7782	9593	10.968 *	19.296	**0.033** **
	1.2	17,375	7782	9.069 *	3.193	0.193
Window-of-Size-2						
Project	**IVersion**	**#InsOWin**	**#InsRWin**	**CD**	**t-Value**	***p*-Value**
prop-1	9	4255	4620	7.540 *	5.422	0.116
	44	8875	3557	51.009 *	32.327	**0.020** **
	92	8177	3527	13.581 *	15.962	**0.040** **
	128	7084	3457	33.327 *	16.404	**0.039** *
	164	6984	3598	0.970	6.025	0.105
prop-2	225	1810	2231	7.216 *	3.661	0.170
	236	4010	1962	1.006	2.307	0.260
	245	4193	1964	34.360 *	16.148	**0.039** **
	256	3926	2307	9.437 *	8.416	0.075

Table 8. *Cont.*

Window-of-Size-2						
Project	**IVersion**	**#InsOWin**	**#InsRWin**	**CD**	**t-Value**	***p*-Value**
prop-3	285	1694	2285	4.693 *	0.626	0.644
	292	3973	2344	0.933	2.208	0.271
	305	4629	2395	11.298 *	8.312	0.076
prop-4	347	2871	2791	39.092 *	21.006	**0.030** **
	355	5662	2854	0.953	11.578	0.055
prop-5	4	3022	4053	1.173	2.366	0.255
	40	7075	3077	29.492 *	16.657	**0.038** **
	85	7130	2998	9.838 *	5.517	0.114
	121	6075	2496	4.893 *	7.311	0.087
	157	5494	2825	1.183	4.408	0.142
prop-42	452	256	192	1.148	2.562	0.237
	453	448	212	4.319 *	14.507	**0.044** **
jm	1	7782	9593	10.968 *	19.296	**0.033** **
	1.2	17,375	7782	9.069 *	3.193	0.193

Table 9. Projects with significant data distributions differences that achieved a large effect size for the window-of-size-1 by adopting Cliff's δ effect size.

Project	**Old Window**	**Recent Window**	**t-Value**	***p*-Value**	**Cliff's δ Effect Size**
prop-1	44	92	24.591	0.026	0.497
	92	128	23.172	0.027	0.484
prop-2	245	256	12.821	0.050	0.437
prop-4	347	355	21.006	0.030	0.490
	355	362	17.890	0.036	0.462
prop-5	40	85	13.823	0.046	0.461
prop-42	453	454	9.811	0.065	0.477
jm	1	1.2	19.296	0.033	0.464

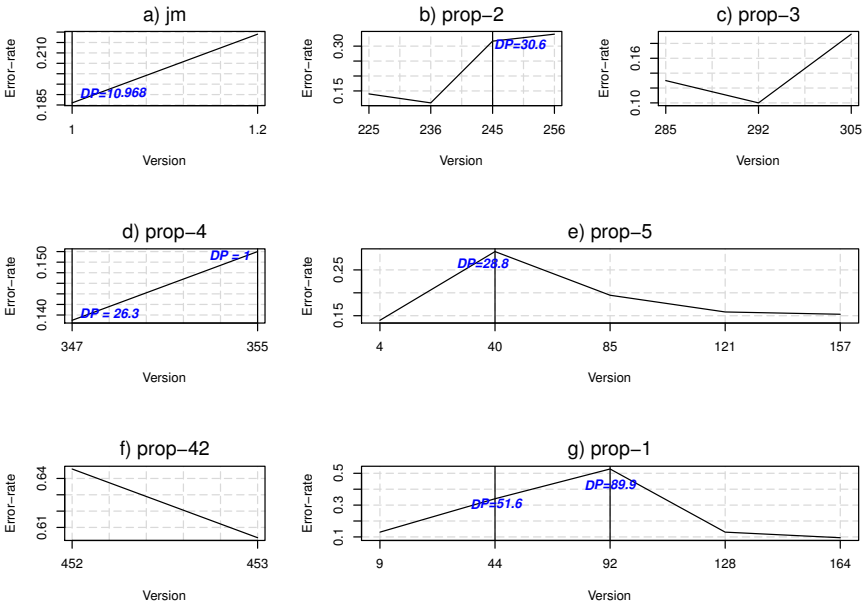

Figure 7. Performance curve and portrait of the error-rate fluctuation of the considered datasets using NB. Significant CD values are inserted and marked by a line across the versions of the chronological datasets.

Based on our proposed CDD framework, the statistical tests, and the empirical experiment (where the data of the versions become available over time) in the context of CVDP, we conclude the following:

- In the chronological defect datasets, our proposed CDD framework identifies CDs by considering the fact that the data of the projects' versions arrives in streaming format.
- The performance of CVDP models is affected by the data distributions in the chronological splitting scenario. The prediction performance fluctuates among the versions for each chronological split approach.
- A trend is observed: when CDs are detected, the error-rate of the affected versions is increased when splitting methods are applied in the experiment.

Our CDD framework is able to spot the CDs in the chronological versions of defect datasets throughout all iterations, as well as the ideal time-point for updating or retraining the prediction models. Our empirical study's statistical analysis revealed that the CD values are statistically significant and that the sampling error is not the cause of the CDs. Therefore, we demonstrated that chronological splitting methods can effectively handle distributions when they are presented in the CVDP chronological splitting format.

5. Discussion

One can speculate on how CD can impede the cross-version prediction performance. Take the following instance as an example. Six variants are included in the dataset prop-1. The model will not be able to detect defects effectively due to the huge distribution discrepancies if it is developed using the data of version 44 as training data to forecast defects for subsequent releases and CD occurs in that version. According to our empirical analysis, the model has to be updated or retrained to improve the prediction performance because there is a large difference between the data from versions 44 and 92, which causes CD in that data distribution and lowers the model performance. In order to examine RQ1 by formulating it, we propose addressing CD in the versions of the defect datasets to forecast defects in cross-version contexts.

Furthermore, a key aspect may be that the window size of the training set changes over time in the context of CVDP. We configure RQ2 for observing the trends in CD detection while varying the window size. The window size is managed by the chronological split approach; this is helpful in processing the data in chronological order when they come in streaming format, as suggested by Lokan and Mendes [36]. Moreover, the RQs are answered by taking the guidance of the Software Engineering Body of Knowledge (SWEBOK V.3.0) [55]. The acquired knowledge from RQs could help to enrich the software professionals for building more reliable software systems.

5.1. RQ1: Is There Any Data Distribution Difference across Software Versions That Degrades the Prediction Performance?

The CDD framework is proposed to answer this question. CDD uses a CD detection method to determine when the distribution differences significantly affect the streaming data, resulting in poor prediction performance. We employ statistical testing, specifically the chi-square test with Yates's continuity correction, to establish the statistical significance of the CD values and to validate that they do not result from a sampling error. Furthermore, in our pair-wise comparisons, the computation of effect sizes using Cliff's δ reinforces the meaningful and practical significance of the identified CDs.

The results from the experiments show that there is a data distribution difference across the project versions that degrades the prediction performance. Concerning chronological defect datasets, we observe that the CDs are identical for all three of the adopted chronological split approaches based on the CD detection procedures. The statistical significance test also identifies uniform CD values, except for the splitting approach of window-of-size-2 for the prop-1 project. The observations in this study are consistent with the PAC learning theory [40], showing that exponential increases in the error-rate of cross-version prediction

models indicates the existence of CD in streaming data, which degrades the prediction performance of the associated models.

5.2. RQ2: Can Insights Be Gained by Observing the Trends of CD Detection Results While the Window Size Varies?

We address RQ2 by observing the results of our experimental study. In the approach of window-of-size-1, the training set is small. The number of versions in the training set, which is used to predict the defects for the subsequent version as it changes over time, always remains one. We observe that the prediction performance curve fluctuates in the cross-version prediction models, meaning that the prediction performance deteriorates. In our approach, CD flags alert the user to significant differences in distribution where the prediction performance degrades. Thus, software quality teams can take necessary steps for improving CVDP. Moreover, by triggering the CD flags, the CDD framework helps to detect CD, and advises users on when to update or re-train models for improving CVDP. Thus, the CD detection results improve the defect-prediction performance.

To observe how the learning curve behaves when the window size is increased, we perform window-of-size-2, where the number of versions in the training set is two, and window-of-size-n, where all available completed versions are used, for the training set. For these approaches, which have sufficient data to train the models, the latest completed version of the project is again used as the test set. We again observe that the learning curve fluctuates over time. The CDs are identical in the same training set among the three splitting approaches. Furthermore, the same CDs are statistically significant in the same training sets, even though the size of the window varies over time for all of the chronological split approaches. For all of these approaches, the CD flags are triggered by the same prediction models, helping the user update or re-train the models to enhance the prediction results in CVDP settings.

To further investigate and sharpen our conclusion on which splitting approach is appropriate, we conducted the Nemenyi post hoc test [56], a popular approach in the SDP literature, to examine the statistical similarity between splitting approaches by using the critical difference (CrDi).

In Figure 8 (using pf), the CrDi is portrayed by a bar and connects the approaches that are statistically similar. Using the computed value of CrDi = 0.70704, we connect the statistically similar approaches. We note that these splitting approaches are not statistically different from one another.

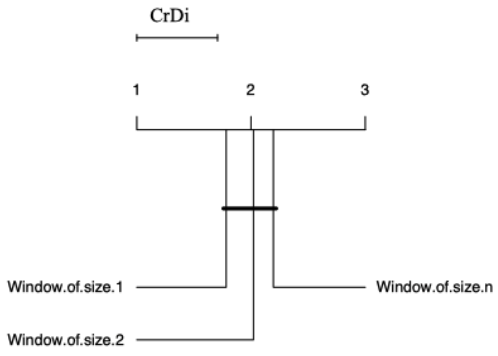

Figure 8. Comparison of pf results of the chronological splitting approaches using the Nemenyi test, with 95% confidence on the cross-version defect datasets.

In summary, in a comprehensive empirical comparison of chronological split approaches, we observe the following results. A moving-window approach, window-of-size-1, is better for CD detection than the other two splitting approaches as it achieves the largest effect size (i.e., $\delta \geq 0.427$). An additional robustness check of our results, the Nemenyi post hoc test, refines our conclusion over which splitting approach should be considered for CD detection. We note that the adopted splitting approaches are not statistically different

from one another. As software projects change over time, new versions are continually released, and a moving-window approach could be helpful in managing the data for CVDP. For conducting experiments in practice and for identifying CD in streaming data, any of the chronological split approaches could be reasonable to use for CVDP, depending on the resources available to a specific software developer.

6. Threats to Validity

This section describes the potential limitations of our empirical study.

External Validity—The concern for external validity revolves around extending the applicability of outcomes obtained through experimental analysis. In our analysis, we chose cases that adhere to the chronological sequence of the software development cycle. We employed the chronological split method to address scenarios involving different software versions. Alternate choices for conducting experiments in cross-version defect prediction (CVDP) are possible. The empirical approach adopted for this study does indeed influence the results of defect detection in the context of version changes. Importantly, our cross-version defect detection (CDD) framework hinges on the presence of class labels within the testing data. Detecting version changes without these labels falls outside the scope of this empirical study, despite its significance as a noteworthy limitation in practical applications.

Internal validity—We conducted our empirical study in chronological defect datasets with a sufficient number of chronologically ordered versions. However, this has been neglected in the literature on CVDP. Motivated by studies of data-stream mining [18], we considered a single base learner, NB, to monitor the error estimates and observe the learning trends. Menzies et al. [35,44] found NB to be the best predictor in defect prediction studies. With respect to the validity of the classification model, it is obvious that the learner in this experiment is a potential bias. To test generalizability, we evaluated the prediction performance using two measures, error-rate and pf, which are widely considered valid measures.

Conclusion validity—Conclusion validity scrutinizes the efficacy of the data analysis procedure in terms of its utility. To examine the effectiveness of the data analysis procedure, we employed the chi-square test along with Yates's continuity correction to identify statistically significant version changes (CDs) within the datasets. Effect-size calculation was conducted using Cliff's δ effect size computation method. To compare the different chronological split approaches, we utilized the Nemenyi post hoc test to determine the most suitable approach.

7. Related Work

In this section, we provide an overview of CVDP. In existing studies, the attempts to avoid distribution differences between two adjacent versions of a project have mostly been based on module selection techniques.

Xu et al. [30] introduced a framework for chronological video data processing (CVDP) that utilizes a combination of hybrid active learning and kernel principal component analysis (KPCA). In the initial phase of their approach, they carefully selected modules from the current version of their software project and augmented these chosen modules with the prior version to create a robust training set. In the subsequent phase, they employed KPCA to map these mixed training modules and trained a logistic regression model. In a series of experiments involving 31 versions across 10 software projects, their aim was to mitigate the distribution disparities between prior and current software project versions by adding crucial modules to the prior version. However, this method has certain limitations. It necessitates substantial effort in labeling selected modules and preparing an appropriate training dataset. Furthermore, the removal of labeled modules from a current software project version can result in a significant loss of valuable information. Another drawback is the need to convert metrics back to datasets for the estimator, which can influence the estimation results during model construction, as noted by [11].

Another work by Xu et al. [9] employed a two-stage process for training-subset selection, with the subset modules chosen from the most recent version. To reduce the burden of labeling the modules of the software projects, they minimized the distribution differences by eliminating the labeled modules from the previous version at the cost of losing some crucial information from that version. Modules from project versions were eliminated in both trials [9,30] in an effort to reduce distributional discrepancies and provide an appropriate training set. This inevitably resulted in the loss of data containing details on both of the most recent and earlier iterations of the software projects. The performance of such models suffers from these information losses.

Yang and Wen [11] investigated the ridge and lasso regression models in CVDP for sorting modules according to defects. They used 41 versions of 11 projects and focused on defect numbers instead of defect categories. Instead of data quality, they targeted the multicollinearity problem, which arises when there are strong correlations among the predictor variables. Their findings confirmed the existence of multicollinearity in the datasets. To prevent the loss of valuable information about the distribution as a result of deleting metrics from the datasets, as found by Xu et al. [9,30], they refrained from the process of variable selection. However, their study [11] did not take into consideration the distribution differences between the versions, which is the most important element to consider when detecting changes between versions to enable the building of effective prediction models.

Shukla et al. [57] tackled the challenge of multi-objective optimization within the context of CVDP. They developed prediction models using multi-objective logistic regression (MOLR) for minimizing the cost (regarding classification and lines of code (LOC)) and maximizing the effectiveness of the model. They found that the distributions of adjacent versions of the same project shared the same distribution parameters. Taking into consideration the Chidamber and Kemerer (CK) metrics [58] from 41 versions of 11 projects in the PROMISE repository, they found that MOLR outperformed four single-objective algorithms in terms of cost-effectiveness.

Bennin et al. [10] conducted a comparative study where they examined 11 density-based prediction models, specifically focusing on the Norm (Popt) effort-aware indicator. This investigation encompassed both inner-version defect prediction (IVDP) and CVDP scenarios. This study took into account various factors, including dataset size, the cost-effectiveness of the models, and validation techniques. Through pairwise comparisons of 25 open source projects, they identified K-star and M5 as the best performers in the CVDP context.

Xu et al. [31] conducted a comparative study in which they evaluated a method called dissimilarity-based sparse subset selection (DS3) against three baseline methods. These baseline methods included the ALL method, as well as the subset selectors Peter Filter [59] and Turhan Filter [60]. Their investigation revealed that the distribution differences negatively impacted the prediction performance of all four models. In an experiment involving 40 cross-version pairs originating from 15 projects, Xu et al. utilized DS3 to create representative modules from the prior version. These modules were then combined with the current versions to construct a new training set. This approach was employed to mitigate the distribution disparities between two consecutive versions by removing labeled modules.

Furthermore, the removal of modules from the prior version can impede the development of an effective prediction model. However, the work of Xu et al. [31] did not show any significant improvement when all of the modules of the previous version were considered (using ALL as a baseline method). Even prediction models built on a subset of modules are perishable, due to information loss caused by removing essential modules. The result of this is to deteriorate the prediction performance of CVDP. The outcome of Xu et al.'s experiment represents a dissenting opinion from that argued by [57].

Bernstein et al. [61] pinpointed that the prediction accuracy variation between SDP models occurs due to changes in temporal features. In attempting to find the latent

relationship between data features, they concluded that the prediction accuracy varies because of the differences among features in the bug generation process. Subsequently, Ekanayake et al. [14] extended the research of Bernstein et al., finding that the models become unstable because of the changes in distribution. They also proposed that concept drift occurs in software-defect datasets, proposed an accuracy-estimation approach in [15], and attempted to identify the most stable prediction models.

In contrast to [11,57], we examine whether the changes in data distribution in a project's versions lead to the degradation of the prediction performance, resulting in untrustworthy prediction models. In three recent studies, Xu et al. [9,30,31] attempted to alleviate the distribution differences in chronological defect datasets by using training-subset selection techniques for improving SDP models. In these approaches, however, the module-selection process involves more effort to process the data and may lead to the excessive loss of useful information, resulting in poor SDP models. Instead, we take all available version data into account, and the versions are considered to arrive in chronological order. Prior researchers [14,15,61] asked why the prediction accuracy fluctuates across SDP models and highlighted the notion of concept drift in defect datasets. However, these researchers did not consider streaming data, did not try to find the significant changes in the data distributions of chronological defect datasets, and did not recognize CD across versions of the same project. In advancing the field of CVDP research, the essential capability to recognize CD becomes pivotal. This identification enables researchers and practitioners to undertake the necessary measures to enhance the prediction performance in cross-version scenarios.

The studies of Gangwar et al. [12,13] introduced an approach known as a pair of paired learners called "PoPL" within the realm of SDP, aimed at addressing CD. The primary objective is to improve the prediction performance beyond what the most successful paired learner methods have achieved in recent times. In a related study, Kabir et al. [62] investigated the performance of class-rebalancing techniques to observe the performance of drift detection and reduction for SDP. The PoPL approach developed by Gangwar et al. [12,13] could be one of the considerable methods for accessing CD for CVDP. We kept this work as one of the our future works.

8. Conclusions

Concept drift (CD) is the phenomenon wherein dataset characteristics gradually shift over time, adversely affecting the performance of predictive models in software engineering (SE), such as those used for cost estimation and defect prediction. In this study, we introduce a sophisticated CD detection framework designed to pinpoint instances of CD within software-defect datasets. Our investigation entails a comprehensive empirical analysis covering 30 versions drawn from seven distinct software projects. These versions exhibit evolving data characteristics as they progress over time. To maintain data integrity and structure it into two-window-based data distributions arranged in chronological order, we employ a chronological split approach. Ensuring the statistical significance of the identified CDs is paramount, and to this end, we leverage Yates's chi-square test. Renowned for its effectiveness in handling two-window-based data distributions, this test confirms that the observed CDs are not a product of mere sampling error. Furthermore, we gauge the practical significance of these CD occurrences by calculating Cliff's δ effect size. This research underscores the critical significance of recognizing and addressing concept drift within software-defect datasets. The occurrence of concept drift is linked to exponential increases in error rates across various software versions. Furthermore, within the chronological splitting scenario, the predictive performance of data distributions exhibits fluctuations.

Therefore, the identification of CD in data distributions is a must while using CVDP for building reliable prediction models. Furthermore, we anticipate that our proposed framework and methodologies will provide valuable assistance to researchers and practitioners in the field of defect prediction. These contributions aim to foster the development

of more robust and adaptable predictive models. As software systems continue to evolve, the capacity to detect and adapt to CD becomes increasingly vital. This study represents a significant stride in that direction.

The study is specifically tailored to software cross-version datasets, making it challenging to extrapolate the proposed framework's applicability to other domains. As a future research direction, it would be advantageous to assess the framework's performance on more diverse datasets, thus expanding its potential utility.

Additionally, it is crucial to acknowledge that the effectiveness of the proposed framework is closely tied to the availability of labeled data. This reliance on labeled data could potentially restrict its applicability in situations where such data are scarce or entirely unavailable.

Author Contributions: Conceptualization, M.A.K.; Methodology, M.A.K.; Software, M.A.K.; Writing—original draft, M.A.K.; Writing—review and editing, A.U.R., M.M.M.I., N.A. and M.L.B.; Supervision, N.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jin, C. A training sample selection method for predicting software defects. *Appl. Intell.* **2022**, *53*, 12015–12031. [CrossRef]
2. Rathore, S.S.; Kumar, S. An empirical study of ensemble techniques for software fault prediction. *Appl. Intell.* **2021**, *51*, 3615–3644. [CrossRef]
3. Gong, L.; Jiang, S.; Bo, L.; Jiang, L.; Qian, J. A Novel Class-Imbalance Learning Approach for Both Within-Project and Cross-Project Defect Prediction. *IEEE Trans. Reliab.* **2019**, *69*, 40–54. [CrossRef]
4. Lu, J.; Liu, A.; Dong, F.; Gu, F.; Gama, J.; Zhang, G. Learning under Concept Drift: A Review. *IEEE Trans. Knowl. Data Eng.* **2019**, *31*, 2346–2363. [CrossRef]
5. Vinagre, J.; Jorge, A.M.; Rocha, C.; Gama, J. Statistically robust evaluation of stream-based recommender systems. *IEEE Trans. Knowl. Data Eng.* **2019**, *33*, 2971–2982. [CrossRef]
6. Schlimmer, J.C.; Granger, R.H. Incremental learning from noisy data. *Mach. Learn.* **1986**, *1*, 317–354. [CrossRef]
7. Turhan, B. On the dataset shift problem in software engineering prediction models. *Empir. Softw. Eng.* **2012**, *17*, 62–74. [CrossRef]
8. Dong, F.; Lu, J.; Li, K.; Zhang, G. Concept drift region identification via competence-based discrepancy distribution estimation. In Proceedings of the 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, China, 24–26 November 2017; pp. 1–7. [CrossRef]
9. Xu, Z.; Li, S.; Luo, X.; Liu, J.; Zhang, T.; Tang, Y.; Xu, J.; Yuan, P.; Keung, J. TSTSS: A two-stage training subset selection framework for cross version defect prediction. *J. Syst. Softw.* **2019**, *154*, 59–78. [CrossRef]
10. Bennin, K.E.; Toda, K.; Kamei, Y.; Keung, J.; Monden, A.; Ubayashi, N. Empirical Evaluation of Cross-Release Effort-Aware Defect Prediction Models. In Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), Vienna, Austria, 1–3 August 2016; pp. 214–221. [CrossRef]
11. Yang, X.; Wen, W. Ridge and Lasso Regression Models for Cross-Version Defect Prediction. *IEEE Trans. Reliab.* **2018**, *67*, 885–896. [CrossRef]
12. Gangwar, A.K.; Kumar, S. Concept Drift in Software Defect Prediction: A Method for Detecting and Handling the Drift. *ACM Trans. Internet Technol.* **2023**, *23*, 1–28. [CrossRef]
13. Gangwar, A.K.; Kumar, S.; Mishra, A. A paired learner-based approach for concept drift detection and adaptation in software defect prediction. *Appl. Sci.* **2021**, *11*, 6663. [CrossRef]
14. Ekanayake, J.; Tappolet, J.; Gall, H.C.; Bernstein, A. Tracking concept drift of software projects using defect prediction quality. In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, Canada, 16–17 May 2009; pp. 51–60. [CrossRef]
15. Ekanayake, J.; Tappolet, J.; Gall, H.C.; Bernstein, A. Time variance and defect prediction in software projects. *Empir. Softw. Eng.* **2012**, *17*, 348–389. [CrossRef]
16. Nishida, K.; Yamauchi, K. Detecting Concept Drift Using Statistical Testing. In *Proceedings of the Discovery Science*; Corruble, V., Takeda, M., Suzuki, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 264–269.
17. Ren, S.; Liao, B.; Zhu, W.; Li, K. Knowledge-maximized ensemble algorithm for different types of concept drift. *Inf. Sci.* **2018**, *430–431*, 261–281. [CrossRef]
18. Nagendran, N.; Sultana, H.P.; Sarkar, A. A Comparative Analysis on Ensemble Classifiers for Concept Drifting Data Streams. In *Soft Computing and Medical Bioinformatics*; Springer: Singapore, 2019; pp. 55–62. [CrossRef]

19. Lu, N.; Zhang, G.; Lu, J. Concept drift detection via competence models. *Artif. Intell.* **2014**, *209*, 11–28. [CrossRef]
20. Liu, D.; Wu, Y.; Jiang, H. FP-ELM: An online sequential learning algorithm for dealing with concept drift. *Neurocomputing* **2016**, *207*, 322–334. [CrossRef]
21. Gu, F.; Zhang, G.; Lu, J.; Lin, C.-T. Concept drift detection based on equal density estimation. In Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, Canada, 24–29 July 2016; pp. 24–30. [CrossRef]
22. Gama, J.; Medas, P.; Castillo, G.; Rodrigues, P. Learning with Drift Detection. In Proceedings of the Advances in Artificial Intelligence–SBIA 2004, Maranhao, Brazil, 29 September–1 October 2004; Bazzan, A.L.C., Labidi, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 286–295.
23. Lu, J.; Liu, A.; Song, Y.; Zhang, G. Data-driven decision support under concept drift in streamed big data. *Complex Intell. Syst.* **2019**, *6*, 157–163. [CrossRef]
24. Babüroğlu, E.S.; Durmuşoğlu, A.; Dereli, T. Concept drift from 1980 to 2020: A comprehensive bibliometric analysis with future research insight. *Evol. Syst.* **2023**, 1–21. [CrossRef]
25. Bayram, F.; Ahmed, B.S.; Kassler, A. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowl.-Based Syst.* **2022**, *245*, 108632. [CrossRef]
26. Kabir, M.A.; Keung, J.W.; Bennin, K.E.; Zhang, M. Assessing the Significant Impact of Concept Drift in Software Defect Prediction. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; Volume 1, pp. 53–58. [CrossRef]
27. Kabir, M.A.; Keung, J.; Turhan, B.; Bennin, K.E. Inter-release defect prediction with feature selection using temporal chunk-based learning: An empirical study. *Appl. Soft Comput.* **2021**, *113*, 107870. [CrossRef]
28. Agrahari, S.; Singh, A.K. Concept drift detection in data stream mining: A literature review. *J. King Saud-Univ.-Comput. Inf. Sci.* **2022**, *34*, 9523–9540. [CrossRef]
29. Madeyski, L.; Jureczko, M. Which process metrics can significantly improve defect prediction models? An empirical study. *Softw. Qual. J.* **2015**, *23*, 393–422. [CrossRef]
30. Xu, Z.; Liu, J.; Luo, X.; Zhang, T. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 20–23 March 2018; pp. 209–220. [CrossRef]
31. Xu, Z.; Li, S.; Tang, Y.; Luo, X.; Zhang, T.; Liu, J.; Xu, J. Cross Version Defect Prediction with Representative Data via Sparse Subset Selection. In Proceedings of the 26th Conference on Program Comprehension (ICPC'18), Gothenburg, Sweden, 27–28 May 2018; ACM: New York, NY, USA, 2018; pp. 132–143. [CrossRef]
32. Menzies, T.; Shepperd M. jm1. *Zenodo* **2004**. [CrossRef]
33. Jureckzo, M. prop. *Zenodo* **2010**. [CrossRef]
34. Wu, F.; Jing, X.Y.; Dong, X.; Cao, J.; Xu, M.; Zhang, H.; Ying, S.; Xu, B. Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, 20–28 May 2017; pp. 195–197. [CrossRef]
35. Menzies, T.; DiStefano, J.; Orrego, A.; Chapman, R. Assessing predictors of software defects. In Proceedings of the Workshop Predictive Software Models, Chicago, IL, USA, 11–17 September 2004.
36. Lokan, C.; Mendes, E. Investigating the use of moving windows to improve software effort prediction: A replicated study. *Empir. Softw. Eng.* **2017**, *22*, 716–767. [CrossRef]
37. Minku, L.; Yao, X. Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models. *Autom. Softw. Eng.* **2017**, *24*, 499–542. [CrossRef]
38. Zhang, L.; Lin, J.; Karim, R. Sliding Window-Based Fault Detection From High-Dimensional Data Streams. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *47*, 289–303. [CrossRef]
39. Iwashita, A.S.; Papa, J.P. An Overview on Concept Drift Learning. *IEEE Access* **2019**, *7*, 1532–1547. [CrossRef]
40. Mitchell, T.M. *Machine Learning*; McGraw Hill: New York, NY, USA, 1997.
41. Lu, N.; Lu, J.; Zhang, G.; de Mantaras, R.L. A concept drift-tolerant case-base editing technique. *Artif. Intell.* **2016**, *230*, 108–133. [CrossRef]
42. Lowery, J.; Hays, M.J.; Burch, A.; Behr, D.; Brown, S.; Kearney, E.; Senseney, D.; Arce, S. Reducing central line-associated bloodstream infection (CLABSI) rates with cognitive science-based training. *Am. J. Infect. Control.* **2022**, *50*, 1266–1267. [CrossRef]
43. Akhtar, A.; Sosa, E.; Castro, S.; Sur, M.; Lozano, V.; D'Souza, G.; Yeung, S.; Macalintal, J.; Patel, M.; Zou, X.; et al. A Lung Cancer Screening Education Program Impacts both Referral Rates and Provider and Medical Assistant Knowledge at Two Federally Qualified Health Centers. *Clin. Lung Cancer* **2022**, *23*, 356–363. [CrossRef]
44. Menzies, T.; Greenwald, J.; Frank, A. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Trans. Softw. Eng.* **2007**, *33*, 2–13. [CrossRef]
45. Bennin, K.E.; Keung, J.; Monden, A.; Phannachitta, P.; Mensah, S. The Significant Effects of Data Sampling Approaches on Software Defect Prioritization and Classification. In Proceedings of the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, ON, Canada, 9–10 November 2017; pp. 364–373. [CrossRef]
46. Bennin, K.E.; Keung, J.W.; Monden, A. On the relative value of data resampling approaches for software defect prediction. *Empir. Softw. Eng.* **2019**, *24*, 602–636. [CrossRef]

47. Bennin, K.E.; Keung, J.; Phannachitta, P.; Monden, A.; Mensah, S. MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction. *IEEE Trans. Softw. Eng.* **2018**, *44*, 534–550. [CrossRef]
48. Amasaki, S. Cross-version defect prediction: Use historical data, cross-project data, or both? *Empir. Softw. Eng.* **2020**, *25*, 1573–1595. [CrossRef]
49. Lokan, C.; Mendes, E. Applying Moving Windows to Software Effort Estimation. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), Lake Buena Vista, FL, USA, 15–16 October 2009; IEEE Computer Society: Washington, DC, USA, 2009; pp. 111–122. [CrossRef]
50. Wilkinson, L. Statistical methods in psychology journals: Guidelines and explanations. *Am. Psychol.* **1999**, *54*, 594. [CrossRef]
51. Carl, E.; Witcraft, S.M.; Kauffman, B.Y.; Gillespie, E.M.; Becker, E.S.; Cuijpers, P.; Van Ameringen, M.; Smits, J.A.; Powers, M.B. Psychological and pharmacological treatments for generalized anxiety disorder (GAD): A meta-analysis of randomized controlled trials. *Cogn. Behav. Ther.* **2020**, *49*, 1–21. [CrossRef]
52. Mensah, S.; Keung, J.; MacDonell, S.G.; Bosu, M.F.; Bennin, K.E. Investigating the Significance of the Bellwether Effect to Improve Software Effort Prediction: Further Empirical Study. *IEEE Trans. Reliab.* **2018**, *67*, 1176–1198. [CrossRef]
53. Kampenes, V.B.; Dybå, T.; Hannay, J.E.; Sjøberg, D.I. A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.* **2007**, *49*, 1073–1086. [CrossRef]
54. Kitchenham, B.; Madeyski, L.; Budgen, D.; Keung, J.; Brereton, P.; Charters, S.; Gibbs, S.; Pohthong, A. Robust Statistical Methods for Empirical Software Engineering. *Empir. Softw. Eng.* **2017**, *22*, 579–630. [CrossRef]
55. Quezada-Sarmiento, P.A.; Elorriaga, J.A.; Arruarte, A.; Washizaki, H. Open BOK on software engineering educational context: A systematic literature review. *Sustainability* **2020**, *12*, 6858. [CrossRef]
56. Ma, J.; Xia, D.; Wang, Y.; Niu, X.; Jiang, S.; Liu, Z.; Guo, H. A comprehensive comparison among metaheuristics (MHs) for geohazard modeling using machine learning: Insights from a case study of landslide displacement prediction. *Eng. Appl. Artif. Intell.* **2022**, *114*, 105150. [CrossRef]
57. Shukla, S.; Radhakrishnan, T.; Muthukumaran, K.; Neti, L.B.M. Multi-objective cross-version defect prediction. *Soft Comput.* **2018**, *22*, 1959–1980. [CrossRef]
58. Chidamber, S.R.; Kemerer, C.F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.* **1994**, *20*, 476–493. [CrossRef]
59. Peters, F.; Menzies, T.; Marcus, A. Better cross company defect prediction. In Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR), San Francisco, CA, USA, 18–19 May 2013; pp. 409–418. [CrossRef]
60. Turhan, B.; Menzies, T.; Bener, A.B.; Di Stefano, J. On the relative value of cross-company and within-company data for defect prediction. *Empir. Softw. Eng.* **2009**, *14*, 540–578. [CrossRef]
61. Bernstein, A.; Ekanayake, J.; Pinzger, M. Improving Defect Prediction Using Temporal Features and Non Linear Models. In Proceedings of the Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting (IWPSE'07), San Francisco, CA, USA, 9 December 2023; ACM: New York, NY, USA, 2007; pp. 11–18. [CrossRef]
62. Kabir, M.A.; Begum, S.; Ahmed, M.U.; Rehman, A.U. CODE: A Moving-Window-Based Framework for Detecting Concept Drift in Software Defect Prediction. *Symmetry* **2022**, *14*, 2508. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.